DOCUMENT RESUME

ED 207 698 PS 012 416

AUTHOR Klahr, David
TITLE Investigation of Pre-School Children's Problem Solving Processes. Final Report.
INSTITUTION Carnegie-Mellon Univ., Pittsburgh, Pa.
SPONS AGENCY National Inst. of Education (ED), Washington, D.C.
PUB DATE 5 Aug 81
GRANT NIE-G-780035
NOTE 88p.

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS *Cognitive Ability; *Cognitive Development; Memory; Models; *Performance Factors; *Planning; *Preschool Children; Preschool Education; *Problem Solving; Puzzles; Reaction Time; Research Methodology
IDENTIFIERS *Developmental Theory; Knowledge Development

ABSTRACT

Preschool children's problem solving processes are investigated in both direct and indirect ways. Direct investigations focus on substantive and methodological issues related to how children solve a few well defined puzzles, such as the Tower of Hanoi and the Tangram. Indirect investigations deal with related issues: U-shaped (or non-monotone) developmental curves, rates of processing, structure-process invariance, and instructional theory. Findings indicate that by the time children reach kindergarten, they appear to have acquired without direct instruction variations on many of the components of mature problem solving strategies. Therefore, attempts to instruct children to be better problem solvers must first make a careful determination not only of the level of their performances, but also of the strategies they use. A methodology involving the characterization of children's knowledge in terms of rules has been developed to facilitate such a determination. The position is taken that U-shaped curves always reflect an artifact of the assessment procedure, do not reflect any interesting underlying processes, and ultimately must be accounted for by general mechanisms of self-modification that are neither constrained nor informed by U-shaped phenomena. The focus on rates, processes, and structures as potential sources of developmental differences maps the domain for further investigations of how children learn to solve problems. (Author/RH)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED207698

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

PS

FINAL REPORT

Grant No. NIE-G-780035

Investigation of Pre-School Children's Problem Solving Processes

David Klahr
Carnegie-Mellon University
Pittsburgh, PA 15213

August 5, 1981

U.S. Department of Education
National Institute of Education

PS012416

The research reported herein was performed pursuant to a grant with the National Institute of Education, U.S. Department of Education. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgement in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official National Institute of Education position or policy.

Table of Contents

1 Introduction 1
2 Research Background 1
2.1 General Orientation 1
3 Problem Solving and Planning 2
3.1 The Tower of Hanoi 3
3.1.1 Results of TOH study 4
3.1.2 "True" planning on the TOH 8
3.1.3 Summary & Discussion of the TOH study 9
3.2 Tangram 11
4 U-shaped Curves 14
4.1 Banking and Calculating: they only look U-ish. 18
4.2 Qualitative and Quantitative Differences in Knowledge Systems 20
4.2.1 Extensive Quantity, Intensive Quantity and Transformations. 20
4.2.2 Quality, Quantity, Specificity and Generality 21
5 Production Systems and Developmental Theory 23
5.1 A Simple Production System 23
5.2 Production Systems for Balance Scale Knowledge 24
5.3 Variable condition elements 28
5.4 Olden Times: State Descriptions 29
5.5 Modern Times: Self Modification 30
5.6 Some Current Examples 31
6 Structure-Process Invariance 34
6.1 Elementary quantification 34
6.2 Alphabetic access 34
6.3 A preliminary description of the model 37
6.4 Experiment I 42
6.4.1 Subjects 42
6.4.2 Materials 42
6.4.3 Procedure 42
6.4.4 Results 42
6.5 Experiment II 44
6.5.1 Subjects 44
6.5.2 Stimulus Materials 44
6.5.3 Procedure 44
6.5.4 Results & Discussion 45
6.6 Description & Evaluation of ALPHA: a model of alphabetic access 47
6.6.1 Representation 47
6.6.2 Processes 49
6.6.3 Parameter estimation and model evaluation 49
6.7 Children's Alphabetic Access 51
7 Instructional Theory 53
8 Summary 54
9 Papers Published During Grant Period 55
10 Professional Activities During Grant Period 56

List of Figures

Figure 1: Three-disk Tower of Hanoi problem. 3
Figure 2: Child seated in front of "Monkey cans" working on a 1-move problem. State 2 to state 1: see fig. 3 5
Figure 3: State space of all legal configurations and moves for 3-can problem. 5
Figure 4: Two protocols and Plan encodings. 6
Figure 5: Proportion of children producing perfect plans. (a) T-end problems; (b) F-end problems. 7
Figure 6: Seven basic pieces of the Tangram. 12
Figure 7: Problems associated with the Tangram. 13
Figure 8: Strategy for placing Tangram pieces. 15
Figure 9: Sequence of piece placement on cat. 16
Figure 10: Protocol on cat. 17
Figure 11: Decision tree representation for four rules about balance scales. (From Figure 1, Klahr & Siegler, 1978) 22
Figure 12: A simple production system with a data base. 24
Figure 13: Production system (P) representations for Models I-IV. D = distance; W = weight. (From Figure 2, Klahr & Siegler, 1978) 24
Figure 14: Decision tree for idiosyncratic rule used by a single child. (From Figure 3, Klahr & Siegler, 1978) 25
Figure 15: Production system for child during decision, feedback, and criterion revision phases of training experiment. (From Figure 4, Klahr & Siegler, 1978) 26
Figure 16: Trace of system shown in Figure 15. (From Figure 5, Klahr & Siegler, 1978) 27
Figure 17: Simple production system changed by hints and practice. 33
Figure 18: RT as function of position and separation (Lovelace, et al). 36
Figure 19: Representation of alphabet in ALPHA. 38
Figure 20: Hypothetical RT pattern from ALPHA. 41
Figure 21: Mean RTs for adult subjects (Exp. 1). 43
Figure 22: Mean RTs for adults (Exp. 2). 46
Figure 23: Frequency of reported entry points. 48
Figure 24: Prediction and actual RTs from "tuned" ALPHA. 52

List of Tables

Table 1: Hypothetical Performance Measures. 18
Table 2: Transformational Category for Operations on 10° C Red Sugar Water. 21
Table 3: The number of "nexts" executed for each probe letter on the **after** and **before** tasks. 50
Table 4: Regression results for ALPHA effort against Exp. 2 data. 51

1 Introduction

This is a final report on NIE grant number NIE-G-780035, "Investigation of Pre-School Children's Problem Solving Processes." This report is organized around several topics related to different aspects of children's problem solving processes. The details of the studies associated with each topic are available in the papers listed in section 9.

2 Research Background

The research program was initially supported by a grant from the Spencer Foundation entitled "Information Processing Models of Cognitive Development." Reports on various segments of the project have appeared over the past several years (Chi & Klahr, 1975, Klahr, 1973a, Klahr, 1973b, Klahr, 1973c, Klahr, 1976a, Klahr & Wallace, 1970a, Klahr & Wallace, 1970b, Klahr & Wallace, 1972, Klahr & Wallace, 1973) and an integrated description is presented in a recent monograph (Klahr & Wallace, 1976). The second phase of the research involved a shift in emphasis from theory formulation at a general level to theoretically guided empirical studies of problem solving and basic processes in young children. This work, to be summarized in this report, was supported by grants from NIE (G-78-0035) and NSF (BNS77-16905).

2.1 General Orientation

Our research falls within the general framework of an information processing approach to the study of cognitive processes and cognitive development. The general paradigm is to formulate information processing models of the child at different levels of knowledge and then to construct a model that explains the change from one level to the next.

Faced with the behavior of a child performing a task or learning how to perform it, we pose the question: "What processing routines and what kinds of internally stored information would a child need in order to generate the observed behavior?" The answer takes the form of a set of rules that can be interpreted by an information processing device, i.e., a computer program. The program thus constitutes a model of the human. Such models are not "pure programming" inventions, for they are constrained by four major psychological criteria: consistency with what we know of the physiology of the nervous system, consistency with what we know of behavior in tasks other than the one under consideration, sufficiency to produce the behavior they purport to model, and definiteness and concreteness.

One distinctive feature of this approach is its emphasis on precision. Since the models are stated in the form of running computer programs, they tend to be much more detailed and explicit than is typically the case. They include empirically testable statements about the functioning of short term memory, the control of attention, and the amount and organization of essential information in long term memory. Although the models tend to be complex, their logical consistency as well as their detailed predictions of behavior in various environments can be directly tested simply by running them.

Once models of different performance levels have been constructed, we can begin to examine the differences among them. Since the model for each performance level is itself quite precise, the nature of the change between one level and the next is better defined than in most other forms of modelling. This is an important point, for a theory of transition between levels can be no better than the model of what is undergoing that transition.

Most of what has just been said about modelling cognitive development applies equally to problems of learning from instruction. The purpose of education is to produce changes in the learner: in the content and structure of the information in her memory, in the processes she applies to that information, and in the procedures for acquiring new information and additional processes. Thus, education can be viewed as an attempt to produce complex changes in an already complex and adaptive system. The more we know about such a system--that is,the better a model of the learner we have--the more effective we can be in our education efforts. The creation of an information processing theory of learning in an "instructional mode" can be viewed as a design problem (Klahr, 1976b). The designer of a learning system must answer questions about when and how learning will occur, and about the effects that learning will have upon the current system. These are, of course, almost the same questions that face the cognitive development theorist.

3 Problem Solving and Planning

The goal of this research is to improve our understanding of how children learn to solve problems. The overall plan is to explore the effects of variations in instructional procedures on children's learning of, and performance on, different kinds of problems. Based on the empirical evidence obtained during these explorations, we construct task-specific information processing models to account for learning and performance in each situation. Research during the grant period focused primarily on one problem, the Tower of Hanoi, and some pilot studies were initiated on a second problem, the Tangram.

3.1 The Tower of Hanoi

The "standard" version of this problem consists of a series of three pegs, and a set of *n* disks of decreasing size. The disks sit initially on one of the pegs, and the goal is to move the entire *n*-disk configuration to another peg, subject to two constraints: only one disk can be moved at a time, and at no point can a larger disk be above a smaller disk on any given peg. A standard three disk problem is shown in Figure 1.

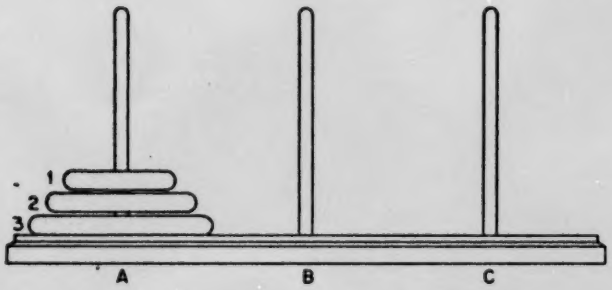


Figure 1: Three-disk Tower of Hanoi problem.

In order to solve this problem you might reason as follows:

> I have to build the stack up from the bottom, which means that I must get disk 3 from A to C, but 2 is in the way, so I'll have to move 2 to B. But if I want to move 2 to B, I must first get 1 out of the way, so my first move will be 1 to C. Now let me reconsider the new configuration. In order to get 3 to C, I still have to move 2 to B, which I can now do. Now in order to get 3 to C I must remove 1 from C, so I will put it on B, and at last I can move 3 to C...etc.

Although there are several other ways to solve the problem, the example shows that even this simple version of the puzzle can tax one's ability to coordinate sequential reasoning, perceptual discrimination, quantitative ordering, and short term memory processes.

For use with preschool children, we modified the task in ways that changed its superficial appearance while maintaining its basic structure.

- Materials. We used a set of nested inverted cans as shown in Figure 2. The cans were modified so that they fit very loosely on the pegs; when they are stacked up it is impossible to put a smaller can on top of a larger can. Even if the child forgets the

relative size constraint, the materials provide an obvious physical consequence of attempted violations: little cans simply fall off bigger cans.

- Externalization of final goal. In addition to the current configuration, the goal -- or target -- configuration was always physically present. We set up the child's cans in a target configuration, and the Experimenter's cans in the initial configuration. Then the child was asked to tell the Experimenter what to do in order to get her cans (E's) to look just like the child's. This procedure was used to elicit multiple-move plans: the child must describe a sequence of moves, which the experimenter then executes.

- Cover story. The problem was presented in the context of a story in which the cans are monkeys (large Daddy, medium size Mommy, small Baby), who jump from tree to tree (peg to peg). The child's monkeys are in some good configuration, the Experimenter's monkeys are "copycat" monkeys who want to look just like the child's monkeys. The cans are redundantly classified by size, color, and family membership in order to make it easy for the child to refer to them. The children found the cover story easy to comprehend and remember, and they readily agreed to consider the cans as monkeys.

- Problem type and difficulty. The standard three disk problem requires 7 moves. We used problems requiring from 1 to 7 moves by systematically using pairs of initial and final states selected from the state-space (Figure 3). For example, state 23 to state 6 can be solved in 1 move, while state 1 to state 15 requires 7 moves. We also varied the type of goal configuration from "towers" (states 1, 8, and 15) to "flats" (e.g., states 3, 6, 10, and 13).

- Planning mode. For each problem, the child told the experimenter the full sequence of proposed moves. The experimenter gave supportive acknowledgment *but did not move the cans*, and then the next problem was presented. The protocol shown in Figure 4 is an example of two perfect 6-move plans.

3.1.1 Results of TOH study

What have we found so far? Children as young as 4 can at least understand the "game", and solve up to 3 move problems. Thus, as far as initial ability to assmilate the rules of a formal problem, even our youngest preschoolers already possess some rudimentary skill. Most impressive, and surprising, is the performance of the 6 year old children. Many of them can reliably produce perfect 6 move plans.

The proportion of subjects in each age group producing correct plans for all problems of a given length is shown in Figure 5a for tower-ending problems and Figure 5b for fat-ending problems. The abscissa in Figure 5 is not overall proportion correct, but rather a much more severe measure: the proportion of subjects with perfect plans on *all* problems of a given length. For example, 9 of the 13 (69%) 6 year olds were correct on all four of the 5 move problems, while only 3 of the 19 (16%) 5 year olds and 2 of the 19 (11%) 4 year olds produced four flawless 5 move plans.

What is striking--given results of previous studies with children on this task--is the absolute level of

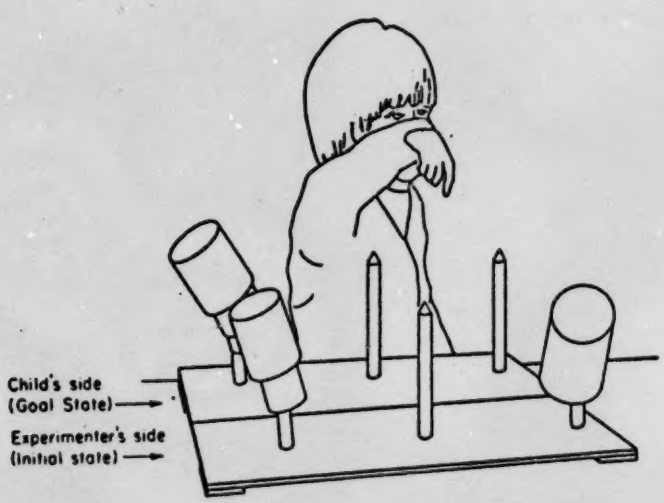


Figure 2: Child seated in front of "Monkey cans" working on a 1-move problem.
State 2 to state 1: see fig. 3

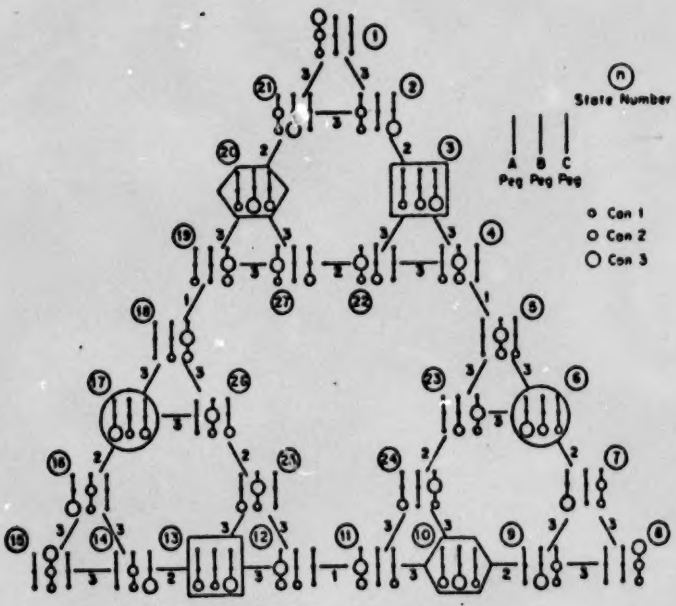


Figure 3: State space of all legal configurations and moves for 3-can problem.

performance. On the T-ending problems, over two-thirds of the 5 year olds and nearly all of the 6 year

Problem 25:	3/21/ _ initial	321/_/_ goal	move	result	
				3/21/ _	(initial)
What you do is you put the daddy (3) . . .					
What you do is you move the daddy (3) over this tree (points to C).					
and move and move the baby . . .					
and then you move the mommy (2),					
wait; where could you move the mommy (2) to ?,					
well first move the mommy (2) on this tree (points to C)			2BC	3/ 1/ 2	
then put the daddy (3) on that tree (points to C)			3AC	_/ 1/32	
and put the baby (1) over there (points to A).			1BA	1/ _/32	
Then how would the mother? . . .					
and after you put the baby (1) over here (points to A)					
you could put the daddy (3) (points to B)			3CB	1/ 3/ 2	
then you could put the mommy (2) over the baby,			2CA	21/ 3/ _	
and the daddy over the mommy.			3BA	321/ _/ _	

Problem 29:	3/ _/21 initial	321/_/_ goal			
				3/ _/21	(initial)
Oh, that. O.K. That's easy.					
Just take the yellow one (3) and put it on there (B).					
Take the (pointing to 2(C)) . . . and take . . . and take, take the ba . . .					
No, take the blue one (2), put it on there (B),			2CB	3/ 2/ 1	
and then, then take the yellow (3)					
and put it on the blue (points toward C, then to B),			3AB	_/32/ 1	
and then take the red (1) one and put it on here (A).			1CA	1/32/ _	
And then take the blue (2) one					
and . . . no, and then . . . and then put the yellow (3) one here (C),			3BC	1/ 2/ 3	
and then put the blue one (2) on the red one,			2BA	21/ _/ 3	
and then put the yellow one on the blue one.			3CA	321/ _/ _	

Can 3	Yellow	Daddy
Can 2	Blue	Mommy
Can 1	Red	Baby

_/	_/	_
A	B	C

Pegs

Figure 4: Two protocols and Plan encodings.

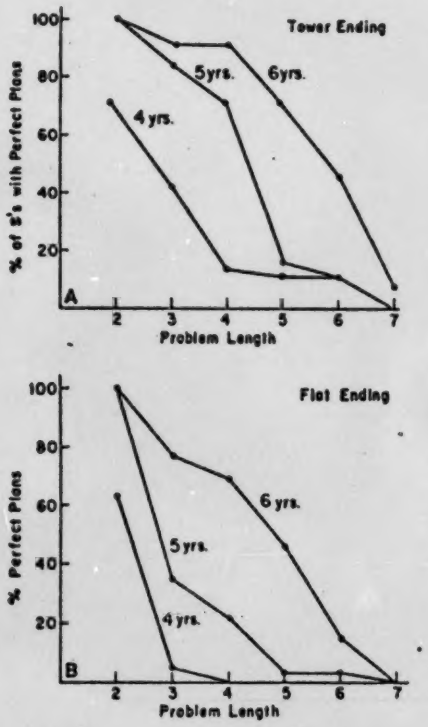


Figure 5: Proportion of children producing perfect plans.
(a) T-end problems; (b) F-end problems.

olds consistently gave perfect 4 move plans, and over half of the 6 year olds gave perfect 6 move plans. Almost half of the 4 year olds could do the 3 move problems. Recall that these plans are verbal descriptions of transformations of hypothetical future states. Furthermore, all intermediate states are different from, but highly confusable with, the two physically present states (i.e., the initial and final configurations).

While the analysis described so far has produced some new information about preschoolers' ability to solve this type of problem, it remains essentially a traditional type of "percentage-of-correct-moves" analysis. Its weakness lies in the fact that it focuses primarily on what children can do rather than on what they know. We felt it was important to go beyond this in order to discover what strategies children use when they generate their responses regardless of whether those responses are correct or incorrect. In order to characterize childrens' inadequate and limited strategies, it was necessary to seek regularities in all of their plans, including the incorrect ones.

For this analysis we constructed a response profile for each child and then matched that profile

against the profiles from a set of plausibly inadequate strategies. The models were constructed in what was essentially a boot-strapping method: based on an informal analysis of the children's responses, we hypothesized a set of partially correct strategies that might generate the observed pattern of moves. Then the models' predictions were compared with the actual moves made by each child on the full problem set, and the best fitting model was chosen as the most characteristic representation of each individual child.

Because of the surprising complexity of this set of models and the many problems against which each model had to be evaluated, we wrote each of the models as a computer program. Each program embodies a particular set of strategic and capacity limitations, and produces a characteristic pattern of correct and incorrect plans on the problem set presented to the children.

Our models ranged from a very simple one that always tried to move the smallest can to its goal peg regardless of the legality of the move, to Simon's (1965) "sophisticated perceptual" strategy. In between those two extremes were models that could attend to one or two obstructors, and which then had to decide whether or not to worry about the obstructor on top of the can that was supposed to be moved or the can that was blocking the goal peg of the desired move.

Next, we obtained, for each of nine models, the characteristic profile that the model generated on the problem set presented to the children. Each of these characteristic profiles was compared with each profile of the six year old children. This profile matching procedure enabled us to accurately (3 models accounted for almost 80% of the subjects' moves) and precisely (since the models are written as computer programs there is no ambiguity as to what move they should have made under any particular circumstances) capture the problem solving strategies used by children on the TOH. The performance of many of the better 6 year olds was captured by a model that had the capacity to search three levels of subgoals; the very best child could solve even our hardest problems, producing a response pattern indistinguishable from the sophisticated perceptual strategy on 7 move problems. At the other extreme, the plan analysis (cf. Figure 5) revealed that the youngest children had difficulty even with 2 move problems.

3.1.2 "True" planning on the TOH

In this study, we gave minimum feedback (recall that children never actually had their plans implemented), in order to accurately access initial competence. This is in marked contrast to the study of "learning by doing" (Anzai & Simon, 1979) in which subjects repeatedly solve the same problem. In such situations, including our own earlier work on the TOH, (Klahr, 1978), subjects quickly discover "subroutines" or "macros" in which the move of a two can stack can be considered

as a single entity, even though it must eventually be unpacked into legal moves. In a subsequent study with 4-can problems, we found evidence suggesting that 6 year olds viewed a goal consisting of 3 cans on one peg and one can on another as two subproblems: - creating the "tower"; and moving the single can to its peg. However, in these 3 can problems, such macro-construction was unusual.

Simply by increaseing the number of objects from 3 to 5 or 6, we can transform the TOH into a problem that is difficult even for adults, and that can best be solved by planning. As the solution path increases in length (to 63 moves for 6 cans) "true" planning, involving the abstraction of detail, becomes necessary. Our present models have no ability to generalize from 2-move macros to the more general notion of a recursive plan for moving stacks of decreasing size. One of our next objectives is to determine the developmental course of this ability and to construct models to account for it.

How and under what conditions will children learn to plan on this task? The models for the strategic variations were initially written as a set of LISP programs (see Appendix A). While LISP was an appropriate medium for generating the 40 move profiles for 10 models, these programs were not well suited for modeling self-modification and change. The strategies are currently being reformulated as production systems in a language called OPS4 (Forgy, 1979) (see Appendix B). These systems of condition - action rules are much more amenable to such analysis, and another objective for our future work is to extend the current state descriptions in the direction of self-modifying systems. (See section 5.)

3.1.3 Summary & Discussion of the TOH study

The results of this study provide clear evidence that by the time children are ready to enter First Grade, they have acquired the rudiments of a non-trivial range of general problem solving methods. Furthermore, they can apply these methods to a novel task. This finding raises two opposing questions: one concerned with why our subjects did so well; the other with why they did not do better.

As for the first: why have other investigators of this problem concluded that young children are capable of no more than trial and error? There are several procedural differences between this and previous studies, but we believe that the most important is our use of very fine-grained levels of differential problem difficulty. While the Plan Analysis indicated that our children were no more successful with the standard 3-disk (7-move) problem, than were Piaget's (1976) or Byrnes & Sptiz's (1979) subjects, a substantial number of them could solve up to 6-move problems. The use of problems whose solution requirements lay between the standard 2-disk and 3-disk problems revealed some previously undetected problem-solving abilities.

It is likely that the externalization of the goal configuration also helped, principally by making it unnecessary to maintain an internal representation of the goal, and thus simplifying the difference detection process. The net effect of the rest of the task modifications (cover story, familiar environment and experimenter, interesting objects, etc.) was to maintain the children's attention long enough to have them make serious attempts to solve the many problems necessary for the profile matching procedure.

Recall that although the 6-year olds did very well up through 5-move problems, the majority of 4-year-old children could not produce perfect plans beyond the 2-move problems (Figure 5), and even their first moves were as likely to be illegal as legal. One might conclude from this that the processes we are studying develop very rapidly between the ages of 4 and 6 years. However, such a conclusion is a bit puzzling when contrasted with the results from investigations of infants' search behavior (Gratch, 1975; Harris, 1975; Piaget, 1954). By the age of 12 months, most children have no trouble setting aside an obstacle in order to reach a desired object if it is visible. And by 18 months, most can use an object as a means to an end, such as reaching a toy on a pillow by pulling the pillow. Thus, the second major question raised by this study is: If children can solve what we have characterized as a two-move problem at 18 months, why do they fail to solve our three-move problems when they are 4 years old?

It is tempting to attribute these discrepancies to "decalage" - Piaget's name for unexpected failure of immediate transfer. For example, the difference between infant search and the poor performance of our youngest subjects on a task requiring verbal solutions could be attributed to *vertical decalage*, i.e., a situation in which "action is more advanced than verbal thought" (Ginsberg & Opper, 1969, p.109). Indeed, in a previous study, we allowed children to move the cans as they solved problems, and the youngest children's performance was somewhat better than in the present study (Klahr, 1978). Of course, the TOH and the infant search tasks differ in many ways other than the verbal - nonverbal distinction; the performance differences may be yet another example of Piaget's *horizontal decalage*. That is, this may be a situation in which "Task contents ... differ in the extent to which they resist and inhibit the application of cognitive structures" (Flavell, 1963, p23). However, the decalage label still leaves open the question as to the nature of the underlying difficulty.

We may begin to answer this question by distinguishing between two intertwined aspects of problem solving: strategies and representations. Thus far, we have focused entirely on the former; in these concluding comments, we offer some speculations about the latter. Throughout the Strategic Analysis, we assumed that the children's encodings were isomorphic to the external display: cans, pegs, positional relations (above and below), size relations, etc. The explanatory power lay entirely in

strategic variations operating on uniform and veridical encodings. While this may be a reasonable approach for the 6-year olds on whom we used it, it is probably not appropriate for the youngest children. The impact on performance of developmental changes in general encoding and attentional processes has been emphasised by Baron (1978) and Klahr & Wallace (1970, 1976). Encoding deficits have been identified as the source of developmental differences in learning (Siegler, 1976), and even with adults, changes in external forms of isomorphic problems produce substantial differences in performance (Simon & Hayes, 1976). We believe that one possible source of difficulty for our youngest subjects was the creation of an internal representation upon which their general problem-solving methods could effectively operate.

General problem-solving methods manifest themselves in rudimentary form by the end of Piaget's "sensory-motor period". They may emerge from the interaction of an "innate kernal" of regularity detectors (Klahr & Wallace, 1976, Chap. 8) and primitive encodings of sensory-motor activity. While much remains to be learned about the developmental trajectory of problem solving methods, we know even less about the development of encoding processes. In future investigations of problem solving by very young children, it will be necessary to provide a more balanced treatment of representational and strategic variation. We will need to direct our attention to the conditions under which task environments are encoded such that they can be appropriately operated on by the rapidly emerging problem-solving processes.

3.2 Tangram

While the analysis of the TOH is well-developed, and has revealed some new and interesting results about children's problem solving and planning skills, it would be premature to make a case for the generality of these results. It is necessary to explore other related problem solving domains, adapting and extending the methodology as required. In this section, we describe a very different problem that we have begun to study: the Tangram. This puzzle, of ancient Chinese lineage, resembles the Western jigsaw puzzle, but it always has the same seven pieces, which are arranged to make a large number of different shapes. The seven basic pieces are shown in Figure 6 and some problems are shown in Figure 7 (taken from a widely available book on the puzzle, (Elffers, 1976)).

The problem needs no motivating "cover story": simply presenting the pieces and the outline of the figure to be built, and asking "Can you fit these all in here?" or "Can you build this from these?" is sufficient to engage preschoolers. Problem difficulty can be varied by varying the number of pieces, and the extent to which their unique contours are revealed in the figure contour. For example, the large square in Figure 6 is one of the most difficult problems (of course it is presented without the benefit of internal contours), while the various "running men" in Figure 7b are relatively easy.

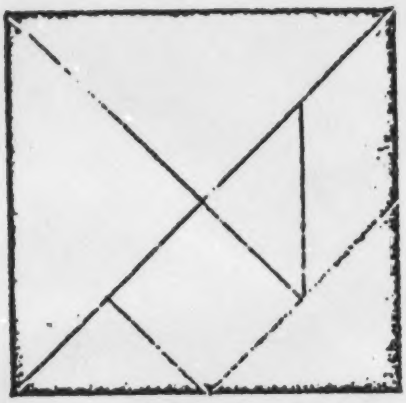


Figure 6: Seven basic pieces of the Tangram.

The tangram provides a good example of the vagueness with which instructional goals related to problem solving and planning are often stated. Tangrams are frequently used as a "manipulable" in primary grade math instruction. One major distributor of educational materials (Creative Publications) suggests using tangrams "to help students learn the concepts of shape, congruence, similarity, perimeter and area." It is clear that the tangram does require simple shape and size recognition and discrimination abilities as well as minimal competence at mental rotation and translation. However, our basic interest is in the *planning skill* that it can be used to reveal. Should one select an area and then search for a piece to fill it, or should one select a piece, and then try to find a suitable location? Which area or piece is most (or least) constrained? When is it clear that an error has been made? How should one recover from an error?

Although the tangram involves moving objects from one physical location to another, it is formally unlike many familiar sequence constrained problems. In such problems, the major task is to determine the optimal - and, in some cases, unique - sequence of operations that transform the initial state into the final state. By contrast, there is no unique or even optimal sequence of piece placements in the tangram. There are no moves that constitute necessary subgoals. All that matters is the ultimate assignment of pieces to locations. In this sense Tangrams are similar to cryptarithmetic puzzles in which addition problems are stated in terms of letters, and the problem is to assign the numerals, 0 - 9, to the ten letters such that a correct addition occurs. Two of the best known examples are

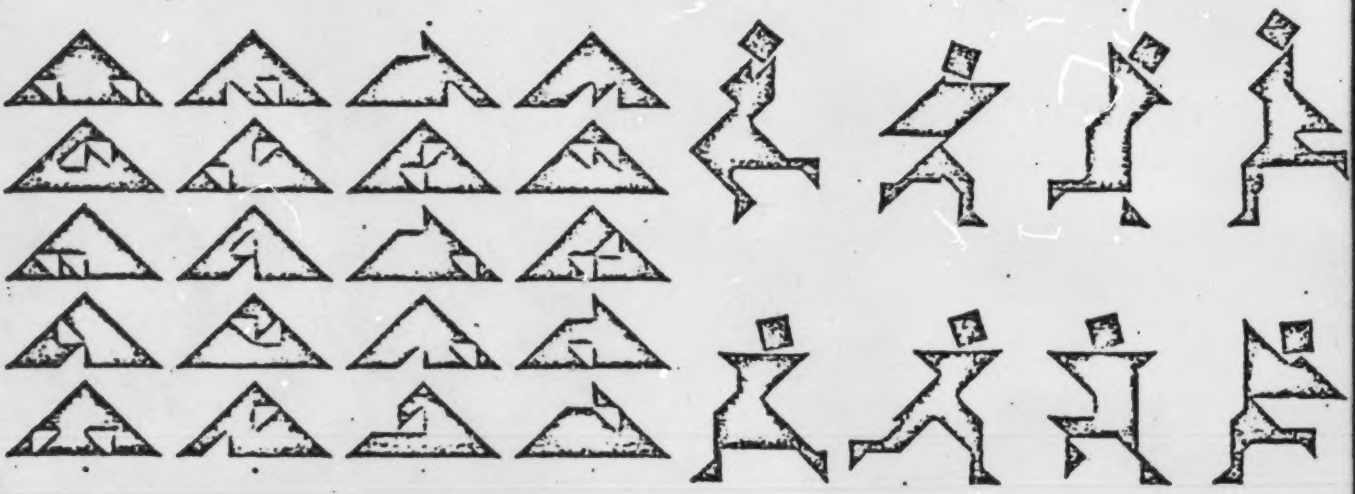


Figure 7: Problems associated with the Tangram.

DONALD	SEND
GERALD	MORE
ROBERT	MONEY

For the first of these, an additional hint is typically provided in the form of one of the assignments : D = 5.

Adult performance on cryptarithmetic problems has been studied extensively by Newell & Simon (1972). Their analysis led to the first use of production systems as a theory of the control structure of the human information processing system. The crux of the solution process consists of tentative assignments of digits to letters, followed by a computation of the effects and further constraints of that assignment. For example, on the first problem, since D = 5, one can compute that T = 0, and that R is odd, since the two L's plus the carry require it. Furthermore, R is limited to 1, 3, 7 or 9, since 5 is already assigned. Further processing limits R to 7 or 9, because in the leftmost column, D + G = R, and we already know that D = 5. In this manner, the constraints proliferate, with incorrect assignments ultimately producing contradictions, at which point the problem solver must back up to the bad assignment and correct it.

Similarly, in the Tangram, early assignments of pieces to places constrain subsequent ones. Bad assignments may not be detected immediately and contradictions (manifested as areas unfillible by remaining pieces) must be backed up to the point of error. Although they have provided extensive

information on adult problem solving abilities, cryptarithmetic problems have never been used with young children, because they require some basic arithmetic operations that are beyond the children's competence. Tangrams provide a vehicle for studying many similar strategic abilities, while building on a basic form matching operator that is readily available to preschoolers.

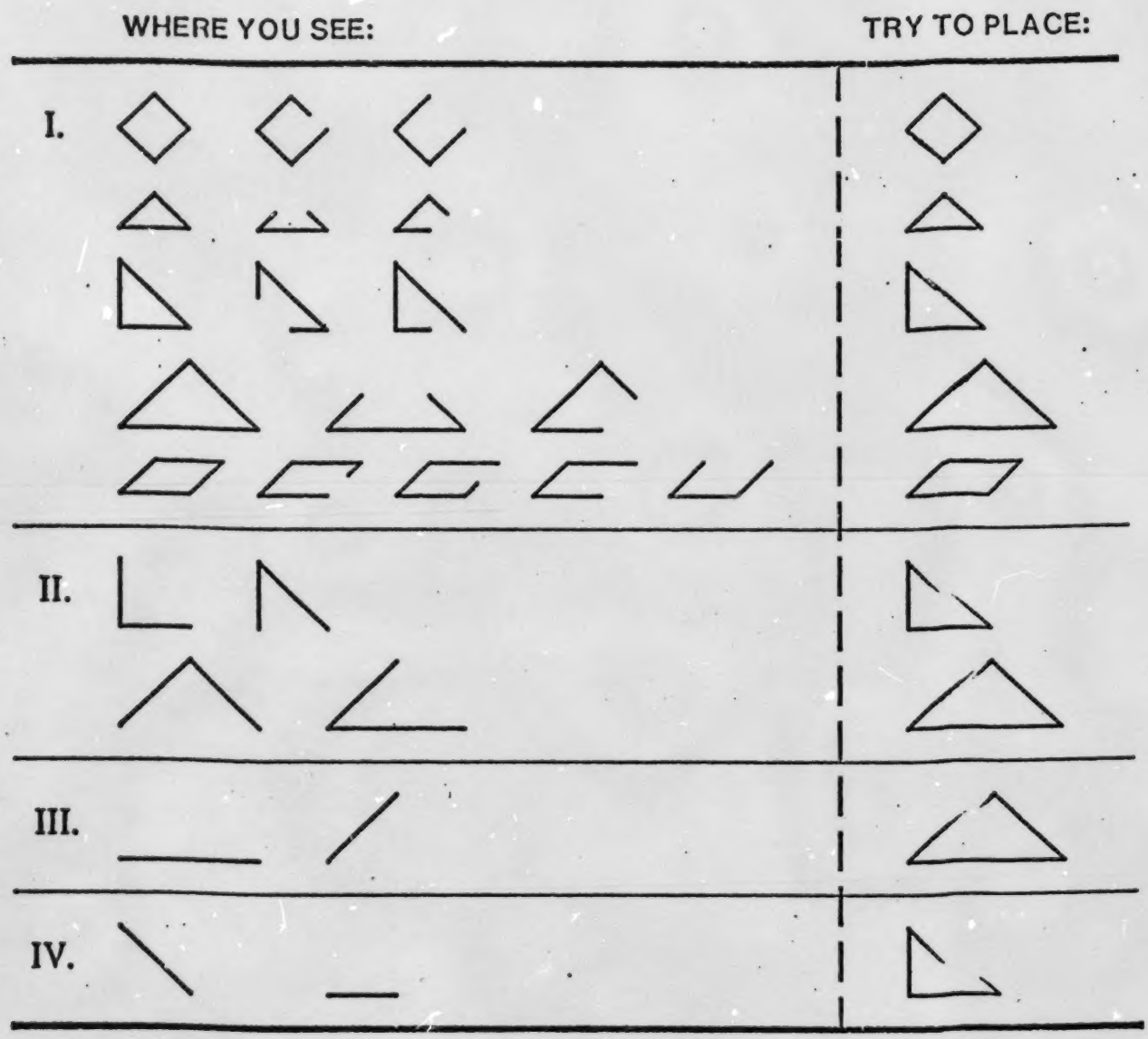
There are three related exploratory investigations of the tangram, concerning young children's performance, adult performance, and a formal analysis of tangram solution strategies. In our pilot studies with 5 year olds, we presented 2-piece problems. Children first had to make, and attempt to justify, a prediction about whether or not the pieces we provided could be used to construct the form presented. Then they were asked to actually solve the problem. We found that 5 year olds have no difficulty understanding the basic task, and that they can solve most of our 2-piece problems.

Our formal task analysis led to the first order strategy shown in Figure 8. The six rules are stated as very general perceptual productions, with an implicit ordering of rule application. The first four rules say, in effect, that pieces should be placed wherever there is unique and minimally ambiguous contour information. (For example, the small triangles and the square would be placed first in Figure 7b.) Rule 5 is much more difficult, for it requires a perceptual test that determines uniqueness of location of a remaining piece. Using this kind of strategy representation, we generated a set of problems at three levels of difficulty: easy problems have all their pieces placed by rules 1 to 4; medium problems require rules 5 and 6 as well; and hard problems cannot be solved by the strategy, either because it leads to incorrect placements or to ambiguous situations.

These problems were then presented to adults instructed to solve the problems while providing a concurrent verbal protocol. [A complete protocol from a very easy problem is shown in Figures 9 and 10.] Preliminary analysis indicates high correlations between our classification of problem difficulty and several performance measures. The protocols also provide a rich source of information about aspects of planning not yet included in our simple task analysis, such as error recover backup, and macros.

4 U-shaped Curves

Issues related to information-processing models of cognitive development were explored during the grant period. One important issue is whether or not empirical demonstrations of the existence of U-shaped growth curves are of importance to developmental theory. In many domains children appear to first perform at a particular level and then, with development, to perform less well, followed ultimately by an increase beyond the initial level. Such "U-shaped" developmental curves have



V. IF <PIECE> OTHER THAN A SMALL TRIANGLE FITS IN ONLY ONE <LOCATION>,

THEN PUT <PIECE> IN <LOCATION>.

VI. IF <EDGE> OF <PIECE> EXACTLY MATCHES <EDGE> OF <LOCATION>,

THEN PUT <PIECE> IN <LOCATION>, ABUTTING THE EDGES.

Figure 8: Strategy for placing Tangram pieces.

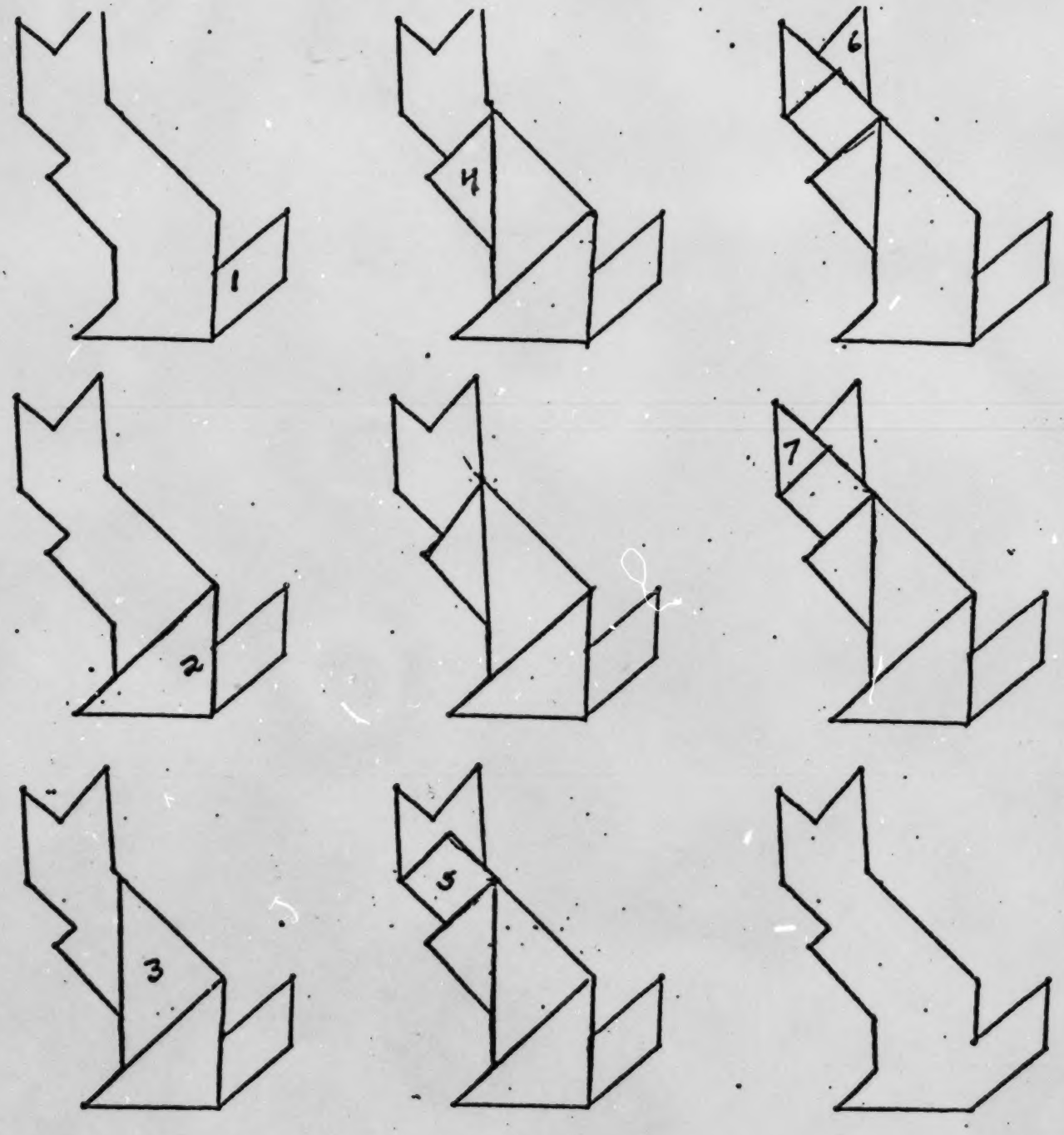


Figure 9: Sequence of piece placement on cat.

00010 S: well
00020 I can almost tell for sure
1
00030 that this is a parallelogram right in here
00040 just by the shape of it fits perfectly
00050 uhm
00060 now by the shape of this going up
2
00070 it says to me
00080 it would be a triangle there
00090 now by the shape of this coming up the thing going up like that
3
00100 this means
00110 the big triangle's going to be there
00120 then by the shape of this
4
00130 it says to me
00140 the medium triangle's going to be there
5
00150 now I have an error
00160 so because of this little piece here
00170 it tells me that it can't be the triangle
00180 because it has that little piece left
6
00190 so I know
00200 that it has to be the square
7,8
00220 which leaves me the two triangles fit in

Figure 10: Protocol on cat.

attracted the attention of many investigators in developmental psychology. Based on my analysis of underlying mechanisms I presented an argument (Klahr, 1981) that such curves are not of fundamental interest to developmental psychology. A summary of the argument is provided here.

There is no doubt that one can routinely discover such curves. They provide empirical accounts of the course of growth of some particular cognitive entity, and therefore they are certainly of descriptive interest. However, I will argue that they always reflect an artifact of the assessment procedure, and they must ultimately be accounted for by general mechanisms of self-modification that are neither constrained nor informed by U-shaped phenomena. That is, anybody going about building a self-modifying information processing system will have to include, in order to explain monotone development, all of the mechanisms that might be postulated to account for non-monotone development. Therefore U-shaped curves do not provide any challenge to developmental theory.

4.1 Banking and Calculating: they only look U-ish.

Let me start with two non-psychological examples of what I mean when I say that U-shaped developmental curves are always measurement artifacts. For the first example, consider the organizational development of a bank. In Stage I, we have a simple bank, where everything happens under one roof. Assume that we define the time it takes to get the loan as a measure of banking performance. One can go into this Stage I bank in the morning, talk to the loan officer for a few hours, and show him or her all the relevant papers. While the loan is approved immediately, you have to allow a few days for paperwork. So the Stage I bank gets a performance measure of 2 days (see Table 1).

	Stage			
	I	II	III	IV
Bank (days)	2	1	7	1/10
Calculator (key strokes)	Many	1	5	

Table 1: Hypothetical Performance Measures.

In Stage II, the bank expands and gets more sophisticated. It acquires some computers. Now there

are several branch banks operating under a management policy of decentralized decision making for loans. You go into this Stage II bank, and the bank manager looks you over; she says you can have your loan; the paper work is all done by computer, and you come back at closing time to get your money. Your performance measure for Stage II is 1 day--a 50% improvement over Stage I.

The bank continues to grow. Now it has many more branches and a computer sufficiently powerful to centralize all decision making about loans. The branch manager in this Stage III bank is really just a customer interface now; she no longer has any discretionary power. Now you go into the bank and you discover that the branch manager is constrained by a central policy, and a central computer. It takes you a week to get your loan and the Stage III bank gets a very poor score with respect to your performance measure. Ultimately this bank expands and gets more and more sophisticated. Finally, the bank reaches Stage IV: to get your loan you give it your Social Security number, your employee payroll number, your secret code, your Master Charge number, your Passport number, and your palm print. It does an instant credit check, and drops the cash out in a little till at your feet. Performance for Stage IV: 1/10 of a day.

Since we are measuring the performance of this bank in terms of how long it takes to give you a loan, we discover non-monotone behavioral growth. In fact, instead of being U-ish, this developmental curve is almost W-ish. Of course, if we had a global assessment of the bank's performance, we would see quite clearly that with respect to all its operations, it has shown monotone growth. *The peaks and valleys we see are a consequence of our restricted view.* Even a slight change in the narrow measure concerned with loans would have shown constant improvement. (For example, number of loans granted per week). Moreover, the sensitivity of the loan granting decision rule has been enormously increased through the replacement of the loan officer's rules of thumb by sophisticated risk computation algorithms.

Second example. Suppose you want to measure the ease of computing a variance on a hand calculator that costs $100. Not so long ago, such a machine would have been a "four-function" calculator, (addition, subtraction, multiplication, and division) perhaps with a square root key, and that was it. Computing a variance required a lot of button pushing-- quite inefficient. Just a few years later, a calculator in that price range had a special button: you entered a bunch of numbers and then you pushed one button and it would do the variance for you. This was often called a statistical calculator. Today, calculators in this price range are programmable. You can write your own program, you can save it, or you can load "canned" programs. However the calculator no longer has a variance button. If you want to compute a variance, you have to make several key strokes; you have to indicate that you want to load library program number 306, and then you'll get general purpose

button A to act like a variance button. So with respect to number of key strokes required to compute variances, this calculator first showed a remarkable improvement and then a decline. Once again, *by focussing on a single performance measure in a system that is undoubtedly increasing in its overall capacity and efficiency, we find a non-monotone curve.*

These examples have been chosen to illustrate the arbitrary and artifactal nature of U-shaped performance measures of the growth of complex systems. Certainly children's minds are orders of magnitude more complex than banks or calculators, and yet, with respect to the systems we are assessing, our experimental measures are often narrower than the performance measures just described. Although we may find many cases of U-shaped curves, they can tell us little about developmental processes.

4.2 Qualitative and Quantitative Differences in Knowledge Systems

Perhaps the most determined effort to demonstrate that U-shaped curves are important for developmental theory can be found in the recent work of Strauss and Stavy (1980). In this section, I will respond to several of their central arguments.

4.2.1 Extensive Quantity, Intensive Quantity and Transformations.

In order to solve Strauss' sugar water problems or Siegler's conservation problems, the child must know something about transformations. Children's knowledge about the effects of such transformations must be empirically grounded, rather than inherent in some innate maturation of cognitive structures. Furthermore, they depend very heavily on the child's appropriate characterization of the effect of different types of transformations on different dimensions of the material. The empirical necessity follows from the fact that no transformation is <u>intrinsically</u> either preserving or changing of quantities. As shown in Table 2, the effect of a given transformation can be categorized only with respect to a dimension of interest.

For example, does the act of pouring conserve quantity or not? The answer depends on both what is poured and <u>what is measured</u>. If we pour a little sugar into red sugar water at 10 degrees C., we do not change temperature, amount, height, width, or redness, but we increase sweetness. If we add more of an identical concentration, we do not change temperature, redness, or sweetness, but amount increases, and so does liquid height, but not width (in a rigid container). On the other hand, if we add water, we increase two "extensive" quantities, reduce two "intensive" quantities, and leave one unchanged.

	Pour to another container		Add material		Add more of same 10° C mix
	Same dimension	Different dimension	Sugar	Water	
Extensive dimension					
Amount	T_0[a]	T_0	T_0	T_+[b]	T_+
Height–width	T_0	T_+	T_0	T_+	T_+
Intensive dimension					
Redness	T_0	T_0	T_0	T_-	T_0
Sweetness	T_0	T_0	T_-	T_-	T_0
Temperature	T_0	T_0	T_0	T_0	T_0

[a] T_0 = null with respect to dimension (e.g. old T_0).
[b] T_+ = change in dimension.

Table 2: Transformational Category for Operations on 10° C Red Sugar Water.

Our own account of the acquisition of conservation rules (Klahr & Wallace, 1976) places a very heavy emphasis on the detection of empirical regularities resulting from specific transformations of specific materials in very limited quantitative ranges. However, there is nothing inherent in a particular physical domain that is of any special psychological interest.

4.2.2 Quality, Quantity, Specificity and Generality

Two intertwined dichotomies in the Strauss and Stavy approach are qualitative vesus quantitative changes in rule systems, and development from specific to general or vice versa. Their view is that the true course of development goes from general to specific, and thus a rule system must go through qualitative, rather than mere quantitative changes. According to Strauss and Stavy, Siegler's rules, and, I suppose, their reformulation as production systems (Klahr & Siegler, 1978), exhibit only quantitative change. Thus, they argue, the resultant view of development is from specific to general. This view, according to Strauss and Stavy is incorrect.

What does it mean to characterize knowledge as specific or general? In Figure 11, is Rule IV more specific or more general than Rule I? I believe that Strauss would call it more general because it is correct on a wider range of examples, but its conditions are more specific.

Siegler's rule system for balance scale predictions (Figure 11) exhibits several properties that are of interest. First I think it is clear that all the tests for the simpler rules are included in the more complex

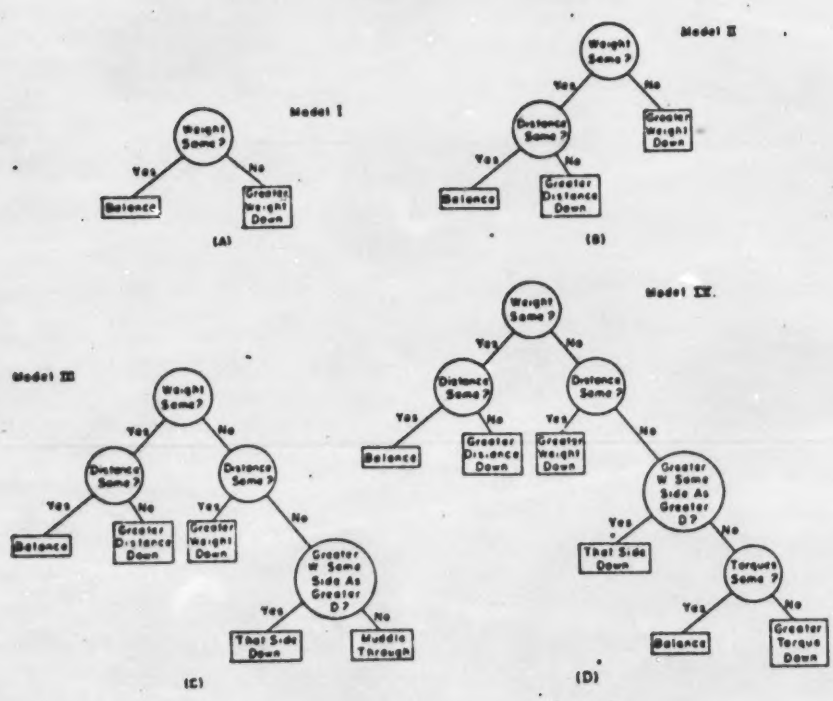


Figure 11: Decision tree representation for four rules about balance scales. (From Figure 1, Klahr & Siegler, 1978)

rules. Secondly, it is clear that the most complex rule (Rule IV), although not including a shift in dimension, does include a qualitative change: that is, to do the torque computation is to invoke a very different set of processes in addition to the simpler ones of comparing weights or distances. Secondly, the torque computation is configural in the sense that the effect of a given amount of weight depends on the amount of distance. Does this qualify as a qualitative change? Notice also that this system does torque computation as a last resort. If there is an easier way to make a decision, the system will make it. In that sense the earlier rules are still manifest in the more complex rules, for with appropriate input Rule IV, will do the same computation as Rule I.

I have been dealing with some notions that are fundamental to our understanding of developmental processes: notions of specificity versus generality, of inclusion versus noninclusion, of inherent contradictions, and of qualitative versus quantitative shifts. I have tried to indicate that, even after the careful attention they have received from many investigators, all of these notions can be rendered vague, imprecise, and self-contradictory. I believe that one problem lies in the medium of our theorizing. Typically we state our developmental theories in words, with an occasional diagram

thrown in for clarity. I believe that information processing models provide a vehicle for theory construction that may enable us to state much more precisely than ever before just what is going on in cognitive development.

5 Production Systems and Developmental Theory

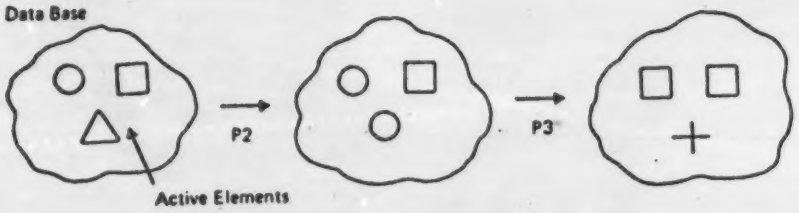
In this section of the report I will try to accomplish two things. First I will describe a non-modifying production system (i.e. a state description), and give an example from an area of interest to cognitive development. Then I will give a brief account of a few self-modifying systems.

5.1 A Simple Production System

A production system consists of a set of productions. Each production--consisting of a condition and an action--has the ability to examine a data base and change that data base contingent upon what it finds there. Figure 12 shows a simple production system. The data base has three active elements--we can think of this as the activated part of long-term memory, as the context for the current processing, as working memory or short-term memory. This data base is examined by a set of productions presumed to exist in long-term memory. These productions are condition-action rules: they say "if you know something about the data base then you can add something else to the data base." The system follows a cycle of recognition and action. In this particular production system the assumption is that once a data base element matches a condition element in a production that fires, then that element is no longer available to fire any other productions unless it is reasserted into the data base. P1 says that if you have a circle and a plus, replace them with a triangle. P2 says replace a triangle with a circle; P3 says if you have two circles, replace them with a square and a plus.

If this production system were to operate on the data base shown here, it would behave as follows. On the first recognition cycle, only P2 would have all of its conditions matched. It would "fire," consuming its input, and adding a circle to the data base. On the next cycle, neither P1 nor P2 would be able to find a complete match, but P3 would be satisfied. It would fire, effectively replacing the two circles with a square and a plus. At this point, none of the productions would be satisfied and the system would halt.

If you take production systems seriously, you have to assume that the human information processing system contains hundreds of thousands of productions, all potentially satisfied in any cycle, but that only a limited subset of the data base is active at any one moment. Many detailed mechanisms that I cannot go into here are described in the growing literature on production systems. This example should give you the flavor, though, of what a production system is.



Productions:

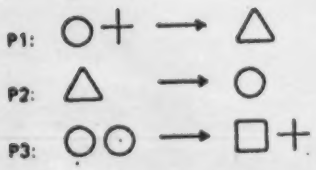


Figure 12: A simple production system with a data base.

```
Model I
        P1: ((Same W) --> (Say "balance"))
        P2: ((Side X more W) --> (Say "X down"))

Model II
        P1: ((Same W) --> (Say "balance"))
        P2: ((Side X more W) --> (Say "X down"))
        P3: ((Same W) (Side X more D) --> (Say "X down"))

Model III
        P1: ((Same W) --> (Say "balance"))
        P2: ((Side X more W) --> (Say "X down"))
        P3: ((Same W) (Side X more D) --> (Say "X down"))
        P4: ((Side X more W) (Side X less D) --> muddle through)
        P5: ((Side X more W) (Side X more D) --> (Say "X down"))

Model IV
        P1: ((Same W) --> (Say "balance"))
        P2: ((Side X more W) --> (Say "X down"))
        P3: ((Same W) (Side X more D) --> (Say "X down"))
        P4': ((Side X more W) (Side X less D) --> (get Torques))
        P5: ((Side X more W) (Side X more D) --> (Say "X down"))
        P6: ((Same Torque) --> (Say "balance"))
        P7: ((Side X more Torque) --> (say "X down"))
```

Figure 13: Production system (P) representations for Models I-IV. D = distance; W = weight. (From Figure 2, Klahr & Siegler, 1978)

5.2 Production Systems for Balance Scale Knowledge

In terms of a state description of a particular level of performance, a production system can be written to embody a set of decision rules a subject might use to accomplish some task. For example, in a recent paper (Klahr & Siegler, 1978), Siegler and I demonstrated the logical equivalence between a

set of simple binary decision trees for partial knowledge about the balance scale, shown in Figure 11, and a set of production systems (Figure 13). At the level of the formal analysis these two representations are equivalent. As the model of the human performance gets more complicated, of course, both representations get more complicated.

Figure 14 shows a decision tree for a child in a training session with the balance scale. She knows a little bit about how the balance scale works, but has only qualitative encodings of weight and distance. This decision tree is now starting to get pretty complicated and it requires a lot of extra interpretation that's not explicit. The production system to do this same task consists of P1 to P8 in Figure 15.

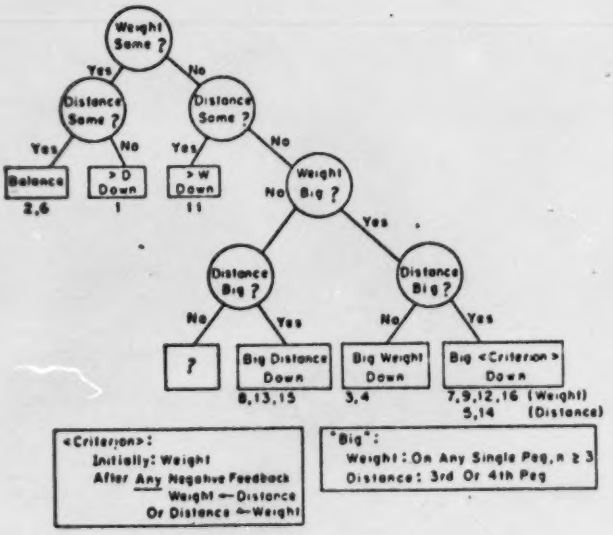


Figure 14: Decision tree for idiosyncratic rule used by a single child.
(From Figure 3, Klahr & Siegler, 1978)

This model can generate a detailed, moment-by-moment, trace of the mental processes that a subject is hypothesized to use as she goes through the task of making a prediction about which way the scale will tip, and then actually seeing it tip, and then trying to revise her hypothesis about whether weight or distance is the dominant criterion. It is clear that in order to capture more of the subject's thinking processes, that is to go beyond the balance scale predictions, and include more of the essestial features of a training condition, we have had to increase the model's complexity.

Figure 16 shows a trace of the model in a Balance Scale training task. The model represents what might be in the subject's active memory after each cycle. Notice there's quite a lot of stuff here that

FILMED FROM
BEST COPY AVAILABLE

```
<dimension.1>:(CLASS weight distance)   <dimension.2>:(CLASS weight distance)

<side.1>:(CLASS left right both)        <side.2>:(CLASS left right both)

<direction>:(CLASS up down level)


P1:((predict) (weight same) --> (made **) (expect both level) say.b)

P2:((predict) (weight more <side.1>) --> (made **) (expect <side.1> down) say.d)

P3:((predict) (weight same) (distance more <side.1>) --> (made **) (expect <side.1> down) say.d)

P4:((predict) (weight more)(distance more) -->  find.big)

P5:((predict) (criterion <dimension.1>)(<dimension.1> big <side.1>)
          (<dimension.2> big <side.2>) --> (made **) (expect <side.1> down)say.d)

P6:((predict) (weight big <side.1>) --> (made **) (expect <side.1> down) say.d)

P7:((predict) (distance big <side.1>) --> (made **) (expect <side.1> down) say.d)

P8:((predict)(<dimension.1>) abs --> ATTEND)


E1:((expect) --> look)

E2:((expect <side.1> <direction>)(see <side.1> <direction>) --> (did **)(see ===> saw)(result correct))

E3:((expect <side.1> <direction>)(see <side.1> <direction>) abs (see) --> (did **)(see ===> saw)
                                                                    (result wrong))


SW1:((result wrong)(criterion distance) --> (old **)(distance ===> weight))

SW2:((result wrong)(criterion weight) --> (old **)(weight ===> distance))

SW3:((result correct)(criterion) --> (old **))


find.big:(OPR CALL) ;returns (weight|distance big left|right), one or two such.

look:(OPR CALL) ; looks for result of balance tipping;
                  returns (see left|right down)

attend:(OPR CALL) ; initial encoding of same or difference on distance & weight;
                    returns (weight|distance same|more left|right)
```

Figure 15: Production system for child during decision, feedback, and criterion revision phases of training experiment. (From Figure 4, Klahr & Siegler, 1978)

the subject is presumed to know. For example, in Cycle 6, she's supposed to know that: she has qualitative encodings of weight and distance, she's currently using a criterion that says weight is important, she made a prediction based on the expectation that the left side would go down, she actually saw the right side go down, she compared those two, and she realized that she was wrong. Now she's got to do something, and the next production that fires (SW2) will be one that says well, if you did all that stuff and your prediction was wrong, you better change the criterion from "weight" to "distance."

FILMED FROM
BEST COPY AVAILABLE

```
        (0003|0020)
Cycle 1
  WM: ((PREDICT) (CRITERION WEIGHT))

  Fire P8: ((PREDICT) (<DIMENSION.1>) ABS --> ATTEND)
       Output from ATTEND (input to WM) ::> (weight more left)(distance more right)

Cycle 2
  WM: ((DISTANCE MORE RIGHT) (WEIGHT MORE LEFT) (PREDICT) (CRITERION WEIGHT) )
     CONFLICT.SET: (P2 P4)
  Fire P4: ((PREDICT) (WEIGHT MORE) (DISTANCE MORE) --> FIND.BIG)
       Output from FIND.BIG (input to WM) ::> (distance big right)(weight big left)

Cycle 3
  WM: ((WEIGHT BIG LEFT) (DISTANCE BIG RIGHT) (PREDICT) (WEIGHT MORE LEFT)
       (DISTANCE MORE RIGHT) (CRITERION WEIGHT))
     CONFLICT.SET: (P2 P4 P5 P6 P7)
     CONFLICT.SET: (P4 P5) AFTER SPECIAL.CASE.ORDER
     CONFLICT.SET: (P5) AFTER WM.ORDER
  Fire P5: ((PREDICT) (CRITERION <DIMENSION.1>)(<DIMENSION.1> BIG <SIDE.1>)
           (<DIMENSION.2> BIG <SIDE.2.) --> (MADE **) (EXPECT <SIDE.1> DOWN) SAY.D)

ssssssssss LEFT down

Cycle 4
  WM: ((EXPECT LEFT DOWN) (MADE (PREDICT)) (CRITERION WEIGHT) (WEIGHT BIG LEFT)
       (DISTANCE BIG RIGHT) (WEIGHT MORE LEFT) (DISTANCE MORE RIGHT))
  Fire E1: ((EXPECT) --> LOOK)
       Output from LOOK (input to WM) ::> (see right down)

Cycle 5
  WM: ((SEE RIGHT DOWN) (EXPECT LEFT DOWN) (MADE (PREDICT)) (CRITERION WEIGHT)
      (WEIGHT BIG LEFT) (DISTANCE BIG RIGHT) (WEIGHT MORE LEFT) (DISTANCE MORE RIGHT))
     CONFLICT.SET: (E1,E3)
  Fire E3: ((EXPECT <SIDE.1> <DIRECTION>)
           (SEE <SIDE.1> <DIRECTION>) ABS (SEE) --> (DID **) (SEE ===> SAW)
(RESULT WRONG))


Cycle 6
  WM: ((RESULT WRONG) (DID (EXPECT LEFT DOWN)) (SAW RIGHT DOWN) (MADE (PREDICT))
       (CRITERION WEIGHT) (WEIGHT BIG LEFT) (DISTANCE BIG RIGHT) (WEIGHT MORE LEFT)
       (DISTANCE MORE RIGHT))
  Fire SW2: ((RESULT WRONG) (CRITERION WEIGHT) --> (OLD **) (WEIGHT ===> DISTANCE))

Cycle 7
  WM: ((OLD (RESULT WRONG)) (CRITERION DISTANCE) (DID (EXPECT LEFT DOWN))
      (SAW RIGHT DOWN) (MADE (PREDICT)) (WEIGHT BIG LEFT) (DISTANCE BIG RIGHT)
       (WEIGHT MORE LEFT) (DISTANCE MORE RIGHT))
```

Figure 16: Trace of system shown in Figure 15. (From Figure 5, Klahr & Siegler, 1978)

Note that this production system deals with two kinds of knowledge that a subject has to bring to bear on a task; not only the formal structure of the problem, but also the demands of the experimental situation. This feature of production systems if of particular relevance to developmental psychology. In almost every area of cognitive development we have discovered that subtle differences in task demands may lead to widely varied performance on the part of our subjects. If we have a modelling procedure that accounts not just for the formal structure of the task, but also for the processing requirements of the experimental situation we might be able to resolve some of the current discussions about why versions A and B of task X lead to such wide differences in performance.

Even more important is the fact that these kinds of models give us the capability to capture the full context of training experiments -- to model some of the micro structure of the developmental process. Indeed, the model depicted in Figures 14 and 15 does just that; it accounts for the subject's response to negative feedback about her prediction.

5.3 Variable condition elements

An important feature of production systems is illustrated by productions E2 and E3 in Figure 15. Their purpose is to detect whether what was expected to occur actually did occur, and the feature of interest is their use of variables in the condition. The first element in E2--(expect <side.1> <direction>)--has two variables in it: <side.1> and <direction>. These are defined at the top of Figure 6 as small classes, any member of which can satisfy the condition element. Thus, if Working Memory contains (expect left down) or (expect right up) etc., the first condition element in E2 will be satisfied. When an element is satisfied, the variable is said to be temporarily bound to the particular value for the rest of the attempt to match the entire condition. If WM contains (expect left down) and (see left down) then E2 will be satisfied. More generally, E2 will be satisfied only when the system "sees" exactly what it "expects."

This ability to perform variable matches and bindings gives production sytems tremendous flexibility to vary their level of specificity, discrimination and generalization. Variable bindings are maintained across the action side (as in P2), so specific information detected on the condition side can be propagated, via the action side, back into Working Memory. This turns out to be a crucial feature of the self-modifying productions (to be described below).

For all their merits, there are many problems associated with the use of production systems. First of all, production systems appear to be very complex to people who are not familiar with them. Secondly, they have many untestable assumptions built into them, and we can only decide whether the whole system makes sense, not whether any single assumption is correct. And they also have bits and pieces of irrelevant mechanism, that is, things that having no psychological validity that are included because they are convenient, or because they represent part of the world that we're not trying to model. Other problems abound, and the literature on information processing models is filled with questions, self-criticism and exciting challenges. (c.f. Haugeland, 1978; Neisser, 1976; Newell, 1970; Pylyshyn, 1978). However, despite all these issues, the area is worth the attention of developmental theorists.

5.4 Olden Times: State Descriptions

A brief history: Several years ago there were production system models for a few tasks of interest to cognitive developmentalists. All of them were based on Newell's (1972, 1973) initial formulation of production system architecture. In 72 Wallace and I (Klahr & Wallace, 1972) did a production system version of class inclusion. We didn't talk about transition at all. Baylor & Gascon (1974) wrote a series of papers on their work with children doing length and weight seriation, and they described different seriation strategies in production system terms. They alluded to the kind of transition process that might be necessary, but they didn't have any model for it. They had only state descriptions. In the same year, Richard Young (1973) completed a dissertation in which he studied length seriation, and accounted for strategic variations with different combinations of productions from a "seriation kit." He argued that children get better on seriation tasks by adding productions specific to disjoint parts of the seriation task.

In 1973 I described a set of production system quantification models in which there was no explicit transition process (Klahr, 1973); rather there was an assertion that one model differed from another in interesting ways, and that the models clarified what the job of the transition mechanism might be. Subsequently, Wallace and I talked about conservation and about how the development of conservation might go (Klahr & Wallace, 1973, 1976). We did directly address the development issue, and we postulated some principles that might constrain the transition processes. We called them consistency detection, redundancy elimination, search for local regularities, and global orientation. But again, there was no running model that actually did the transitions. In other words, all we had a few years ago were some preliminary notions about how production systems might model developmental changes.

Even without explicit transition mechanisms, the "vintage" production systems had much to recommend them as models for interesting developmental phenomena. First of all, production systems are conceived as serious theories of the control structure of the human information processing system. Thus, any particular production system for a particular task setting is a model derived from one of these larger theories, and it integrates many of the psychological principles within it. Secondly, the production systems, as do any simulation models, force a lot of explicitness. They force us to be explicit about how we think parts of the world are encoded, what the encoding process is, how that encoding process generates certain representations, and what kinds of strategies or processes are used by those representations. This explanation, in turn, suggests improved experimental procedures for evaluating our theories (c.f. Trabasso, et al., 1978).

Having done all that, the production system formalism gives us a very clear statement of what state differences might be. Because we can compare two different systems, the job of the transition mechanism is now a lot clearer. Another merit, mentioned above, is that they can include the experimental demands in the same structure as the formal demands of the task, and they allow us to better understand the interaction of the task itself with the general characteristics of the human information processing system.

Almost every current developmental theory emphasizes the centrality of the child's own activity on the learning process. Certainly it is the case that we must understand what it means to engage actively in a cognitive event. But quite contrary to a common critical characterization of information processing models as static and passive (c.f. Neisser, 1976), they are the only theoretical formalisms that actually engage is such activity. They do encode their environments, they do create internal representations, and they do seek matches between what is known and what needs to be done.

5.5 Modern Times: Self Modification

A capability for adaptive self-modification is essential for developmental theory, but until recently, there have been no well specified ideas about how it takes place. What we need is a way to get beyond vague verbal statements of the nature of the developmental process. Perhaps the most important merit of production systems is that they provide a basis for modeling self-modification that goes beyond ambiguous processes such as "assimilation" and "accommodation."

How does self-modification take place in adaptive production systems? What evokes it, and what what are its effects? It is beyond the scope of this report to provide any more than a superficial answer to these questions, but one important fact reduces my reluctance to oversimplify: the systems really do run, and they are available for inspection. Thus any violence I do them can be rectified by a careful reading of the original papers.

In general, all the systems make use of the ability to bind values from Working Memory to the variables in the action elements. Suppose we define x as a variable--x: (class cat dog). Then a production like P1: ((x) --> (saw x)) might match WM: ((dog) (cat)), and produce WM: ((saw dog) (cat)). It would fire again, and produce ((saw dog) (saw cat)). Now consider an action that creates a new production--call it BUILD. A production building production can be included in a production system, and can wait for a certain condition--specified in terms of a mix of constants and variables--before building a new production. For example,

P2: ((saw x)(saw y) --> BUILD ((x)(y) --> (saw x and y))).

If P2 were applied to the WM from the above example, it would produce a new production:

((dog)(cat) --> (saw dog and cat)).

Notice that this new production is less general than P1, since it only fires on two specific values. On the other hand it is more efficient than two firings of P1.

When a self-modifying production system is operating, it need not go from a distinct performance mode to a distinct learning mode. Instead, the performance and the learning productions have equal status with respect to responding to the elements in the data base. When a new production is created, it becomes part of the total set of productions that might fire on the next recognition cycle.

5.6 Some Current Examples

In this section, I will describe four different research projects that are investigating different aspects of self-modifying production systems. Perhaps the clearest examples of self-modification are provided by Anzai's model of learning during a single experimental session (Anzai, 1978; Anzai & Simon, 1979). The system proposes some general mechanisms for learning how to solve a problem during repeated attempts at problem solution. Given the current interest in the potential similarity between micro- and macro-developmental processes (Karmiloff-Smith, 1979), the Anzai & Simon work is of particular interest to cognitive developmentalists.

Pat Langley has worked on a wide range of problems with self-modifying production systems. He started (Langley, 1980) by studying people trying to induce rules for numerical combinations and wrote a production system to account for that induction process. The next development in Langley's work (in press) was a self modifying production system that captures some essential aspects of the scientific discovery process. His system can, given the approriate empirical regularities, induce rules equivalent to Kepler's Third Law, Bode's Law, the Inverse Square Law, and Ohm's Law. None of the induction mechanisms depend on any explicit properties of the physical world. They derive instead from regularity detectors operating on quantitative symbols.

Anderson, Kline and Beasley (1978), within the formalism of Anderson's ACT production system, have built a general model for self-modifying production systems that can account for concept learning, schema abstraction, and some features of language acquisition. They have not just talked about transition mechanisms, they've built them. They have programs that run and do these things. One kind of transition mechanism is a designation production in which a production simply has as its action side the instructions to build another production of a certain form. Another kind of transition mechanism is something they call strengthening: a central part of Anderson's system which determines whether or not a production will fire is how strong it is. Yet another form of transition

mechanism is to build a generalized production, in which some of the specific conditions are weakened in order to make the production more broadly applicable. On the other side of that coin are mechanisms that discriminate under certain conditions. In order to build such a system, Anderson and his colleagues have to specify first exactly how it works, and second the conditions under which these various mechanisms, (designation, strengthening, generalization, and discrimination) will occur. This they have done, and they have been able to account for some of the better known results from the experimental literature in the domains listed.

A particularly interesting example comes from a dissertation by Clayton Lewis (1978). Lewis studied the problem of how an adaptive production system could, by using some simple rules for self modification, demonstrate two effects of practice observed with humans--speed up and Einstellung (Luchins, 1945). The former is evidenced in the robust speed-practice curves in almost every area of human activity; the second is the venerable effect whereby practiced subjects may be less able to utilize a hint or short cut than unpracticed ones. As we shall see, Lewis' work also provides a nice example of how the production system formalization can help to clarify many of the important concepts in cognitive development. The following example is adapted from Chapter 1 of his thesis.

Consider a simple production system that replaces one symbol with another until it reaches a goal. Figure 17 shows several such productions. P1, P3, and P6 replace single symbols, and P4 and P5 each replace a pair of symbols with a different pair. P1 is the stopping rule. The first production system, PS.1, consists of P1, P2, P3, and P4. When PS.1 starts to operate on a data base containing only the letters A and C, the ensuing sequence is AC, BC, DE, and GE. During this sequence, 4 productions fired, and 4 symbol replacements occurred.

Assume that after much practice on this task, a learning mechanism notices that AC invariably produces DE, and a new composite production is formed which directly reflects this constancy. P5 is added to the initial PS.1 system, producing a "practiced" system, PS.2. When the new system runs, it achieves its goal in fewer cycles, and fewer symbol replacements. It has avoided an intermediate state, but although quantitatively different from PS.1, i.e., faster, it is qualitatively unchanged, i.e., no new intermediate states occur, and it arrives at the same final state (GE).

Now suppose a new rule--a hint or shortcut--were somehow given to the initial system and the practiced system. The results are shown at the bottom of Figure 17. If the hint (P6) was provided before practice produced P5, then it would take advantage of intermediate state BC. On the other hand, the practiced system never generates an intermediate state that can satisfy P6, and the hint has no effect. Thus, the production system exhibits the Einstellung effect: shortcuts which can be used before practice are ignored after practice.

P1: G → stop
P2: A → B
P3: D → G
P4: B,C → D,E
} Initial set

P5: A,C → D,E Composite production
P6: B → G hint

Initial System

PS.1: (P1, P2, P3, P4) "performance"
Trace:
Trace: AC $\overset{P2}{\rightarrow}$ BC $\overset{P4}{\rightarrow}$ DE $\overset{P3}{\rightarrow}$ GE $\overset{P1}{\rightarrow}$ stop 4 cycles; 4 replacements

Practiced System with Composite

PS.2: (P1, P2, P3, P4, P5)

Trace: AC $\overset{P5}{\rightarrow}$ DE $\overset{P3}{\rightarrow}$ GE $\overset{P1}{\rightarrow}$ stop 3 cycles; 3 replacements

Hint to Initial System

PS.1H: (P1, P2, P3, P4, P6)

Trace: AC $\overset{P2}{\rightarrow}$ BC $\overset{P6}{\rightarrow}$ GC $\overset{P1}{\rightarrow}$ stop 3 cycles; 2 replacements

Hint to Practiced System

PS.2H: (P1, P2, P3, P4, P5, P6)

Trace: AC $\overset{P5}{\rightarrow}$ DE $\overset{P3}{\rightarrow}$ GE $\overset{P1}{\rightarrow}$ stop 3 cycles; 3 replacements

Figure 17: Simple production system changed by hints and practice.

Practice produces a qualitative change as well as a quantitative one, for the practiced and unpracticed systems obviously differ in their response to the hint, and the use of the hint leads to new intermediate and final stages. This example does not address many important issues, such as the rules under which competing productions are selected, and the exact mechanism of production creation. But it does begin to indicate the way in which issues of importance to developmental theory can be clearly stated.

All of these systems have to specify a set of conditions under which production building processes like generalization, discrimination, designation or strengthening will occur. In all cases, it is possible that the new productions will degrade rather than improve performance. Although the local effects of

self modification are "mechanistic" and "preplanned," the global impact is often unpredictable: it may produce conflicts, inconsistencies and nonmonotone behavioral growth. Whether or not the system gets better or worse depends in part on the circumstances that lead to the self modification, and in part on the subsequent environmental demands on the system.

6 Structure-Process Invariance

Another important theoretical issue is the question of what aspect of the information-processing system develops. In investigations of child-adult differences in information-processing abilities, a distinction is often made between differences in processes -- i.e., in strategies for encoding and accessing information -- and differences in structures -- i.e., in the underlying database upon which those processes operate. In this section I describe two situations in which children and adults appear to have the same underlying processes and structures, but in which adults are nonetheless able to execute their strategies at up to ten times the rate of children.

6.1 Elementary quantification

The first task domain is elementary quantification (Chi & Klahr, 1975). In the experiment, subjects were presented with random arrays of dots, and asked to respond, as rapidly as possible, with the number of dots. Adult reaction times were best fit by two linear regressions. The lower segment (for n = 1-3) had a slope of approximately 50 msec per item, and the upper segment (n = 4-10) had a slope of 300 msec per item. Six year old children had a similar qualitive function, but the lower and upper slopes were approximately 200 and 1000 msec, respectively. The process generating the lower slopes has been termed "subitizing", while the process for numbers greater than 3 is some form of counting or subitizing and adding. Thus, while both children and adults appear to encode and process small numbers in one way and large numbers in another, children's processing rate is 3 to 4 times slower than adults.

6.2 Alphabetic access

The second task is simple alphabetic access. What comes after Q? What comes before H? We asked subjects these sorts of questions in order to learn about how familiar, long lists are stored and accessed in memory, and how the internal representations and the retrieval processes change with time. We focused on the alphabet because it is a common long list, with little explicit structure, learned very early and used throughout life.

Based on our investigations, we have been able to propose a specific internal representation for the

alphabet, a detailed model of the processes used to access the representation, and an estimate of the speed of basic processes of the model.

Three major types of experimental procedures have been used in previous investigations of alphabet storage and access:

1. In Order Decisions the subject must decide whether or not a presented letter pair is in the correct alphabetic order.

2. In the Target Recitation procedure the subject is presented with a pair of letters, and must recite the alphabet (covertly or overtly) from the first letter to the second letter.

3. Forward or Backward Search. In this task the subject must say what comes *nth* after or before the presented letter. For example: "what comes *before* P", or "what comes four letters *after* K."

Regardless of the procedure used, if one looks at reaction time as a function of alphabetic position of the stimulus, two findings consistently emerge: a) At the aggregate level, stimuli at the end of the alphabet tend to require more processing than stimuli at the beginning; b) The RTs are definitely non-monotone, and the fine structure of the RT pattern is similar across a variety of procedures.

Lovelace & Spence (1972) used a forward search procedure. They found an irregularly increasing RT as a function of alphabetic position. The increase from the early portion of the alphabet to the final was substantial. The mean reaction time for the first six letters, A - F, was 890 msec, and for the last six letters, T - Y, was 1180 msec.

Lovelace and his colleagues proposed two possible processes that lead to the increasing RTs. One possibility is that there are lower associative strengths between adjacent letters near the end of the alphabet. These weaker associative strengths lead to longer RTs. That is, it would take longer to do a "next" toward the end of the alphabet than near the beginning. The other possibility is that the interletter associative strengths are equal throughout the alphabet, but there is differential access to particular letters. That is, there might be *preferred entry points* in the alphabet, with fewer such entry points toward the end of the alphabet. This would lead, on the average, to longer search sequences (and higher RTs) for probes near the end of the alphabet.

In order to discriminate between these two possibilities, Lovelace, Powell and Brooks (1973) used the target recitation procedure, varying the number of letters processed after accessing the probe letter. They presented letter pairs with different separations, and the subject's task was to recite the alphabet from the first to the second letter. Figure 18 shows the RT versus the alphabetic position for

separations of 2, 4 and 6 letters. Linear regressions run through these three sets of RTs reveal nearly parallel functions. Lovelace *et al* concluded that the longer reaction times at the end of the alphabet do not come from greater difficulty of doing "nexts"; for if that were the case there would be a fan effect rather than parallel lines. Rather, they come from a greater difficulty of entering the alphabet near the end.

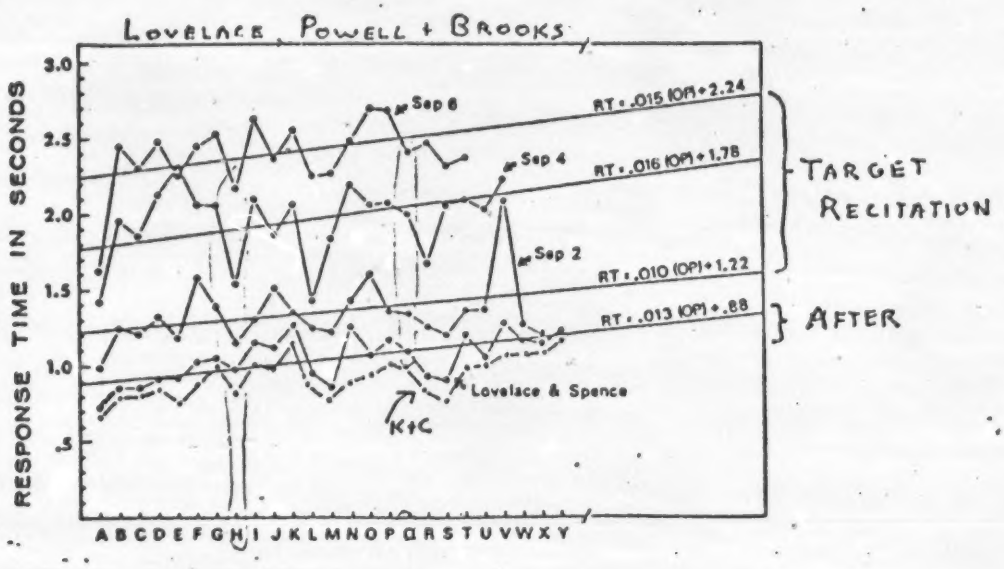


Figure 18: RT as function of position and separation (Lovelace, et al).

Many questions remain concerning the structure and processing of the alphabet.

- Although the Lovelace, *et.al* study supports the notion of preferred entry points, it does not provide any *direct* evidence. The present investigation demonstrates such entry points.

- The model, as stated thus far, is largely intuitive, with no specification of the representation or processes involved. The present paper describes a detailed model, written as a computer simulation, with model parameters estimated from the data.

- The fine structure of the RT patterns has never been accounted for. The present model attempts to predict the RT for each alphabetic position in both forward and backward search tasks.

6.3 A preliminary description of the model

The model, shown in Figure 19, assumes a two level hierarchy consisting of a series of 5 or 6 chunks containing from 2 to 8 letters each. Given a probe, there is a serial, self-terminating search for the chunk membership of the probe, followed by a serial, self-terminating search for probe position within a chunk. Searches for chunks can be bi-directional, searches for position within chunk are unidirectional in the forward direction.

On the **after** task the process works as follows. First, search for the chunk containing the probe; having found the chunk, scan for the probe; when the probe is found do a "next" and output that value. If there is no next, that is, if the end of the chunk has been reached, then get the next chunk and output the first item.

For the **before** task start by searching for a chunk containing the probe item; when the chunk is found, scan for the probe, keeping track of the prior position. When the probe is found, then get the prior item and output it. However, if the probe is at the beginning of a chunk, they there is no prior item. In this case get the prior chunk, scan to the end of that chunk and output the last item.

All of this is shown more concretely in the hypothetical example shown in Figure 20. We assume a segmentation in which the first chunk consists of letters A - G, then H - K and so on, as shown at the top of the Figure. There are two basic times associated with this model. The time to move in either direction at the chunk level is t_1, time to move in the forward direction within a chunk is t_2.

For the **after** task the model would work as follows. After A requires some constant amount of time plus 1 chunk search plus 1 next within a chunk. After B would consist of 1 chunk search plus 2 nexts; After C would consist of 1 chunk search plus three nexts, and so on. Now consider what happens near the end of the chunk boundary. After F requires 6 nexts within a chunk. After G requires an extra chunk search, and 2 extra nexts. After H requires 2 chunk searches, but only 1 next. Continuing this analysis leads to a hypothetical function as shown in the lower curve: the important features are a non-linear increase at the end of a chunk followed by a sharp decrease after the chunk boundary. Note however, the steadily increasing values for local minima.

For the **before** task, analysis is similar. We have used t'_2 to indicate that doing a next and carrying along a prior pointer takes longer than simply doing a next. Before B requires 1 chunk access, and 2 nexts; before C, 1 chunk and 3 nexts. When we get to before G, the chunk boundary, we maintain the linear increment in reaction times; however when we get to before H we have to cross a chunk boundary (from the second chunk to the first), so we have to do 2 chunk searches, followed by a

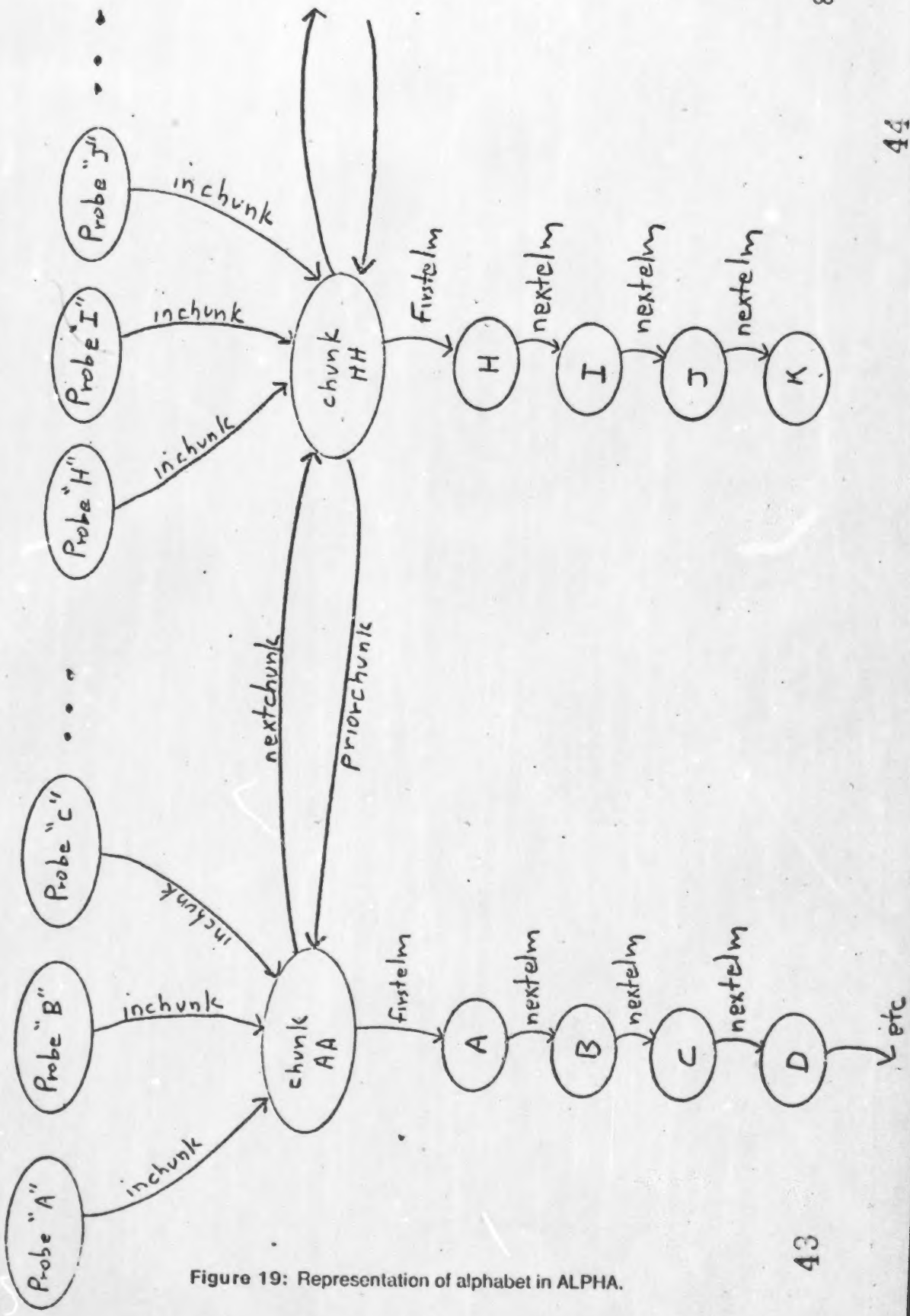


Figure 19: Representation of alphabet in ALPHA.

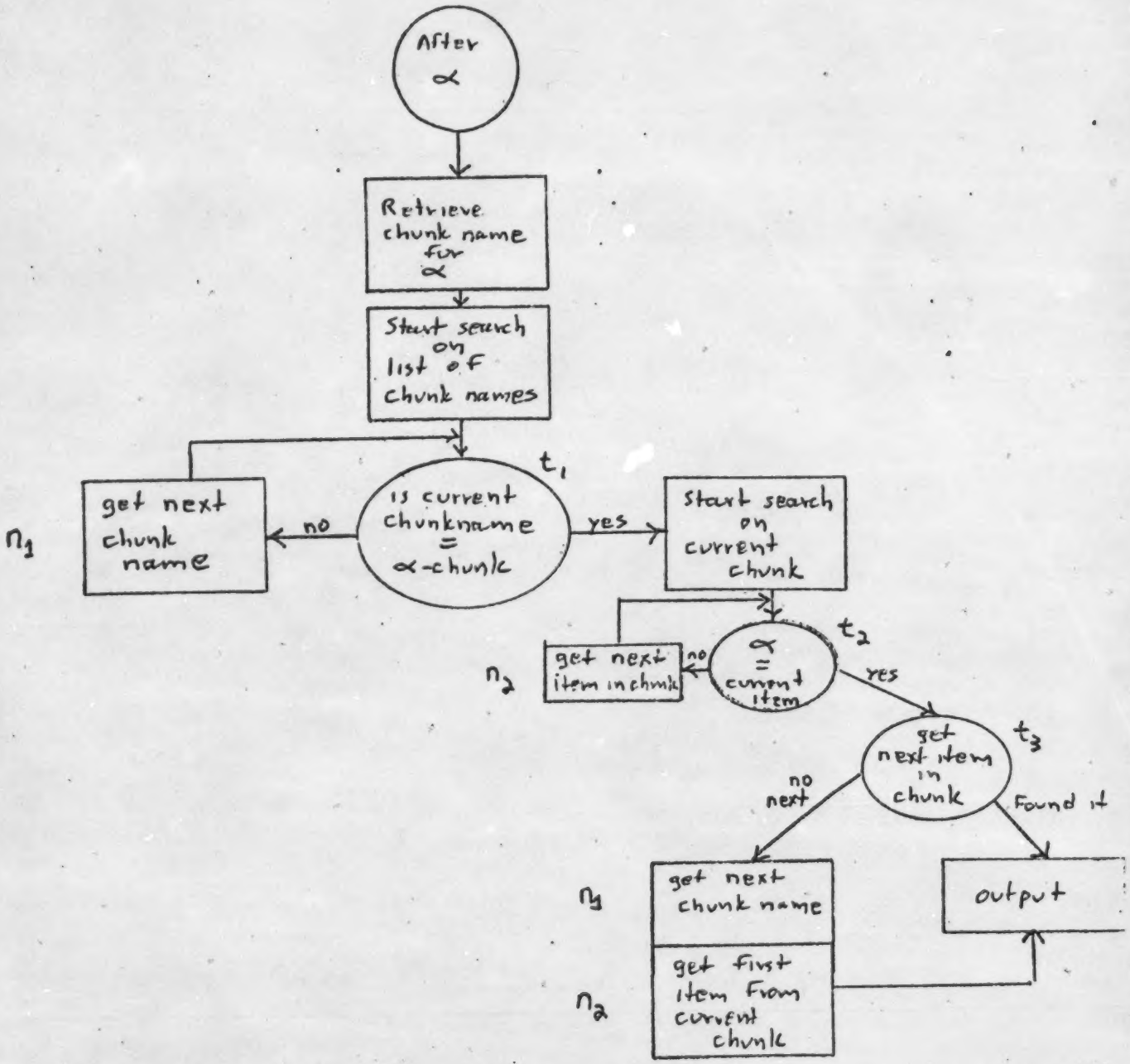


Figure 19a: ALPHA on "after".

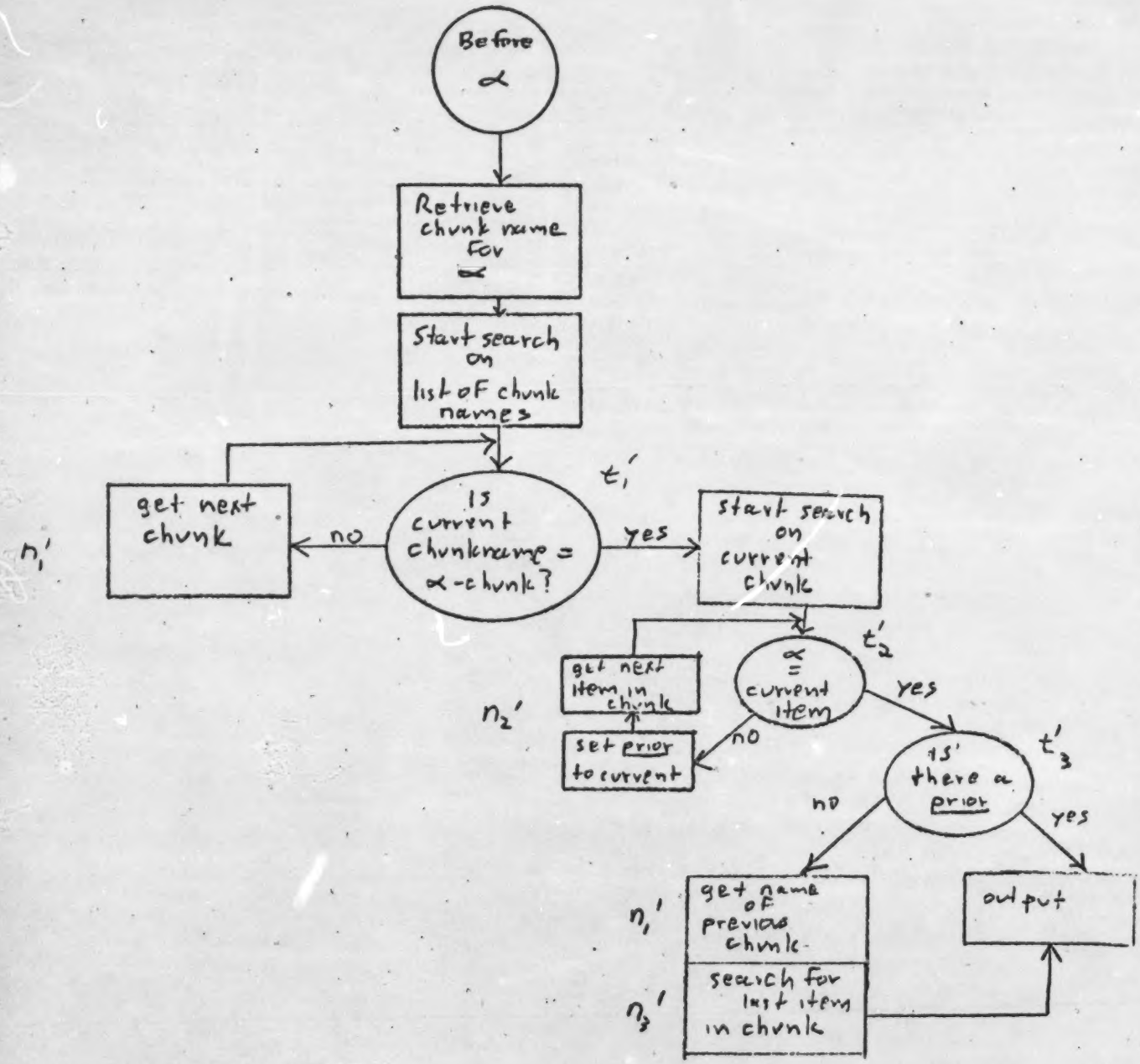


Figure 19b: ALPHA on "before".

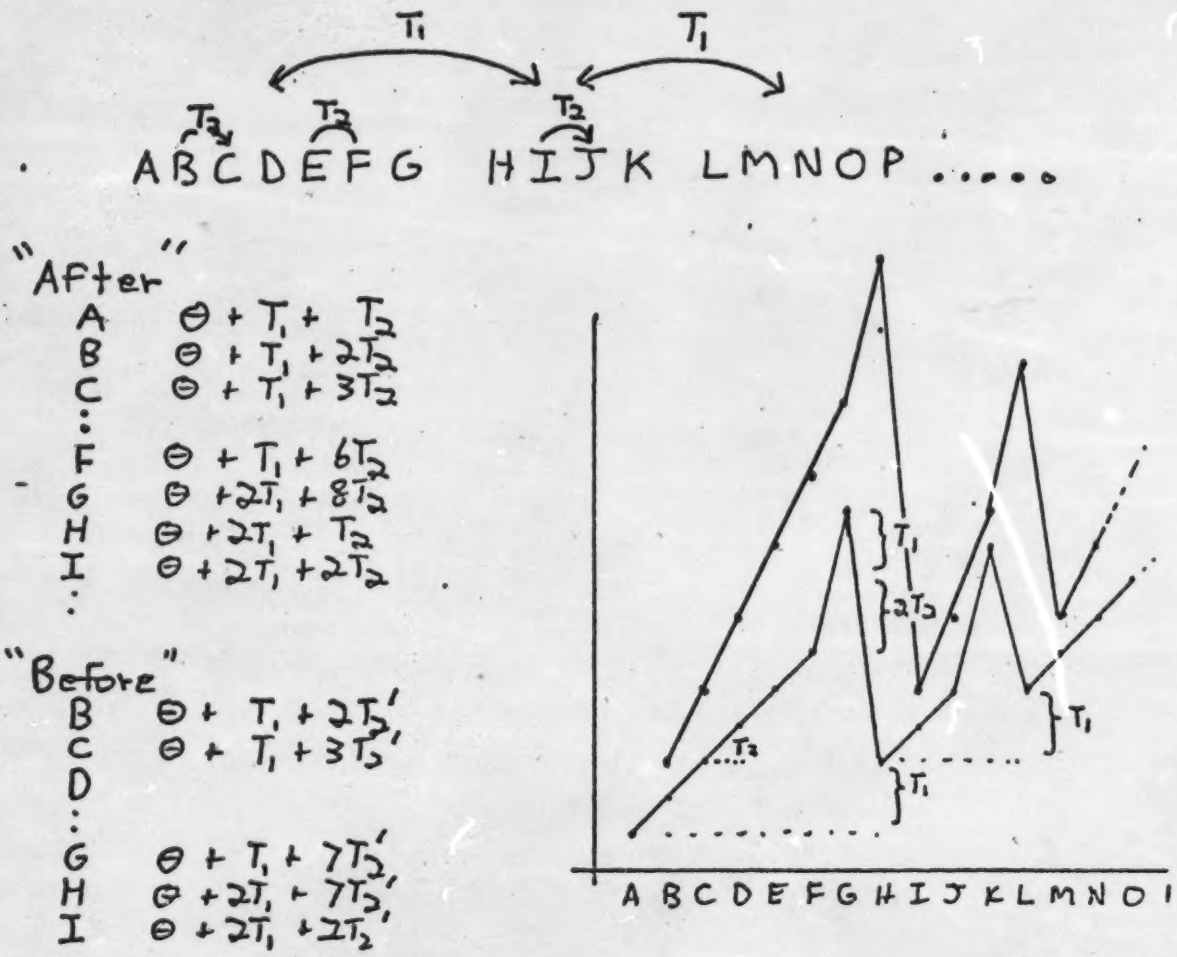

T1
T1
T2
T2
T2
A B C D E F G H I J K L M N O P .....
"After"
A Θ + T1 + T2
B Θ + T1 + 2T2
C Θ + T1 + 3T2
F Θ + T1 + 6T2
G Θ + 2T1 + 8T2
H Θ + 2T1 + T2
I Θ + 2T1 + 2T2
"Before"
B Θ + T1 + 2T2'
C Θ + T1 + 3T2'
D
G Θ + T1 + 7T2'
H Θ + 2T1 + 7T2'
I Θ + 2T1 + 2T2'
T1
2T2
T2
T1
T1
A B C D E F G H I J K L M N O I

Figure 20: Hypothetical RT pattern from ALPHA.

search for the end of the first chunk. Then, when we get to Before I, we don't have to search the entire list, so before I is faster than before H. The salient features of this curve are a non-uniform increase in reaction times, and a local maximum on the **before** task at the beginning of a chunk boundary, followed by a local minimum for what comes before the second item in a chunk.

In summary, the model makes specific predictions about the relationship between local maxima and minima on the **before** and **after** task. The local maxima should occur at the end of a chunk for the **after** task and at the beginning of a chunk for the **before** task. The minima should occur at the beginning of a chunk for the **after** task and at the second element in a chunk for the **before** task. Since none of the previous studies used the **before** task, we conducted an experiment to assess this model, using both the **before** and the **after** task.

6.4 Experiment I

6.4.1 Subjects

Twelve adult subjects from introductory psychology courses participated in this experiment.

6.4.2 Materials

The stimulus letters were all upper case, helvetica-medium, 1.5 in. high, black on a white 5" x 8" card. A voice actuated microphone connected to a Standard timer recorded reaction times to hundreths of a second. E would then record Ss reaction time, change the letter to appear next, and reset the timer to zero. Stimulus onset was subject initiated, following a 500 msec delay.

6.4.3 Procedure

Subjects were instructed on the operation of the T-scope. They were asked to name aloud the preceding letter or the following letter (depending on the condition) as quickly as possible without making any errors. Each S was given three practice trials, and was then given an opportunity to ask questions or clear up any misunderstandings before the experiment began.

Each S received each of two experimental conditions. In the **before** condition, there were five successive presentations of a randomized set of 25 stimulus letters B to Z, for which S was to name the preceding letter. In the **after** condition, there were five successive presentations of a randomized set of 25 stimulus letters A to Y, for which S was to name the following letter.

The order of conditions was counterbalanced across subjects. In the event of an error, E would replace the card in the stack randomly.

6.4.4 Results

For each subject, the median RT (out of five trials) for correct responses to each letter was determined for the **before** and **after** tasks. Figure 21 presents the means (over the 12 subjects) for these median RT's as a function of the alphabetic position of the stimulus letter.

The most striking feature of these curves is their agreement with the predicted relationship between peaks and valleys of the **before** and **after** curves. For example, a local maximum on the **after** curve occurs at G; the local maximum on the **before** curve is at H; while for the **after** curve, H is a local minimum. Similarly, the local max at K on **after** and a local max at L on **before** are associated with the local min on **after** at L. Strong effects here are at the boundary between G and H, the boundary between K and L, and boundary between P and Q. Other boundary effects are a bit weaker, and there are some anomalies toward the end of the alphabet.

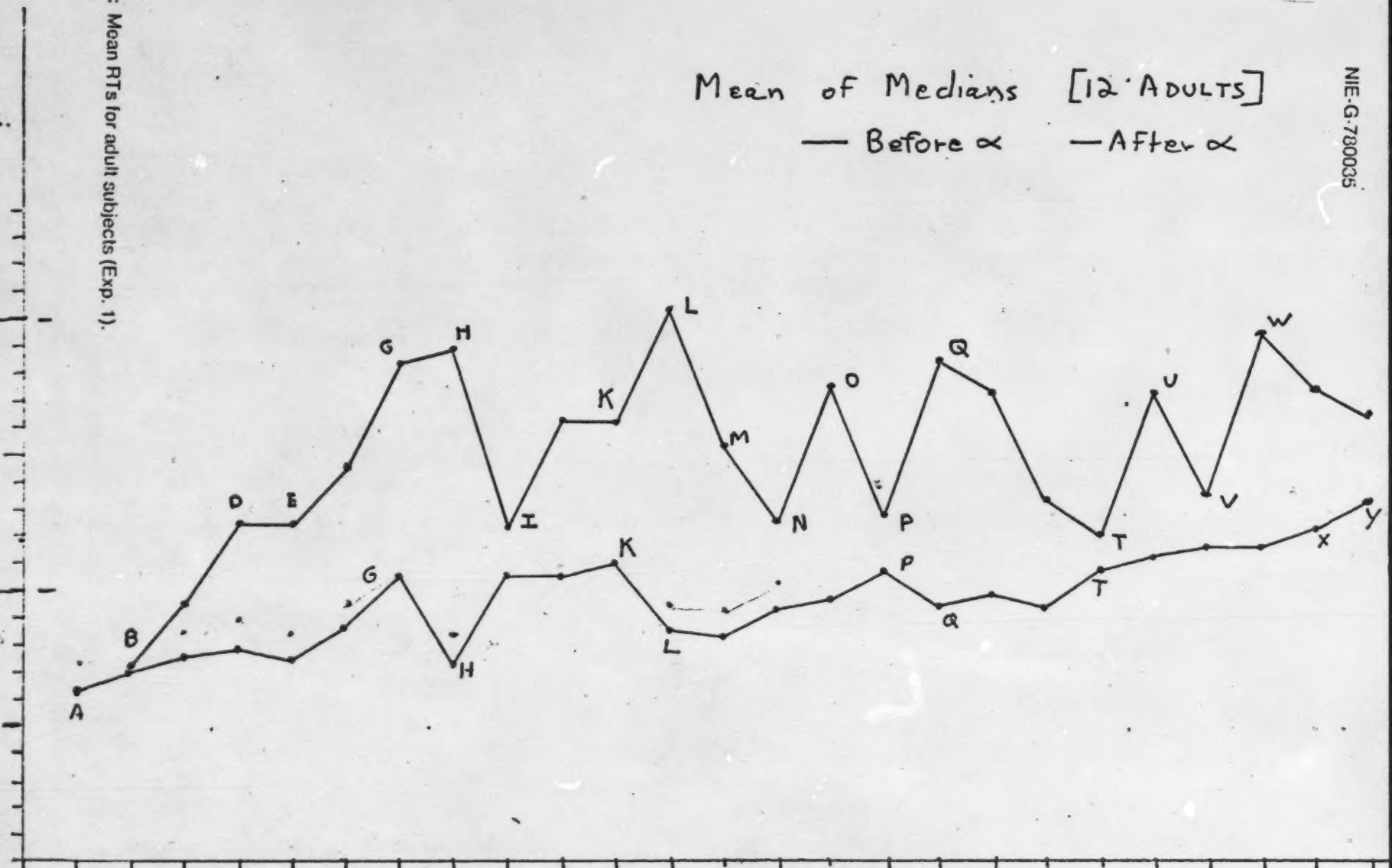


Figure 21: Mean RTs for adult subjects (Exp. 1).

6.5 Experiment II

While these results are consistent with the predictions from the model , the decision about how to segment the alphabet is based on an informal *post hoc* analysis of local extreme points. The fact that this segmentation is consistent with the phrasing in the common nursery school "alphabet song", provides some additional basis for believing it to be correct, but we have no direct independent evidence that it is the segmentation used by our subjects. In the second experiment, we asked subjects to directly report their "entry points", if any. The independent assessment of the segmentation allowed us to perform a much more rigorous evaluation of the model.

6.5.1 Subjects

Thirty students from introductory psychology classes participated in this experiment, half of these being run by each of two experimenters.[1]

6.5.2 Stimulus Materials

Each subject was presented six sets of slides. Each set contained one slide of each of the 26 letters of the alphabet (Artype No. 1407 capitals). Sequencing of the letters within a slide set was random with the restriction that no letter follow the same letter in any two sets. On half the occurrences of a given letter the subject was required to name the preceding letter of the alphabet; on the remaining occasions the task was to name the following letter. (On all six occurences of the letter A the task was to name the following letter and on all occurences of Z to name the preceding letter.)

6.5.3 Procedure

Each subject was seated before a translucent screen in a sound-deadened chamber; the experimenter and all apparatus for stimulus presentation and response timing were outside the chamber. Single letters were back-projected onto the translucent screen, and the subject was to say aloud, as quickly as possible, either the preceding or the following letter of the alphabet. Each stimulus letter was preceded by a warning buzzer, and by one of two different colored lights marked "PRECEDING" and "FOLLOWING" which informed the subject of the type of decision required on that trial. A photocell on the back of the screen activated a Lafayette Model 5721 digital timer when the letter came on the screen. The subject's spoken response activated a voice key which stopped the timer and advanced the projector to an opaque slide. The experimenter initiated each trial manually; the buzzer and light preceded the stimulus letter by approximately 1.5 sec. The stimulus letters were presented at a rate of about 12 per minute.

---

[1]This experiment was designed and run by Professor E.A. Lovelace, at the University of Virginia. Lovelace's work was not supported by funds from this grant.

The subjects were also instructed that once they had responded with the appropriate letter they were then to tell the experimenter what they had done to think of the correct response. These verbal reports were classified into three categories: a) didn't have to do anything, the letter just occurred to me, b) had to covertly recite a specifiable portion of the alphabet, or c) had to do something, but not explicitly described as recitation of a specific portion of the alphabet. Whenever a subject reported covert recitation of part of the alphabet they were asked to indicate the letter at which they began that recitation if possible.

6.5.4 Results & Discussion

Most people were able to maintain very high accuracy levels while operating with a speed set; overt errors of naming the wrong letter occurred on less than 1% of the trials. Voice key equipment malfunctions and other errors account for data lost on about 1.5% of the trials.

For each individual the median reaction time (RT) for correct responses to each letter was determined separately for the **before** and **after** tasks. Figure 22 presents the means of these median RTs as a function of the stimulus letter presented. The data for the **before** task from Lovelace and Spence (1972) are shown by the bottom curve; there is high correspondence between those times and the **after** condition in the present study, r = .81, although the times were both longer and more variable in the present study where **before** and **after** tasks were mixed. The RTs from experiment II are also highly correlated with those from experiment I [r = .78 for **after**, and r = .93 for **before**] even though in Experiment I we used a blocked design on **before** and **after**, while in Experiment II, **before** and **after** trials were mixed.

These relative frequencies of reported necessity to "do something" on **after** trials correlated highly with RTs on those trials (r = .90) and with **after** RTs in the earlier data of Lovelace and Spence (r = .88). For **before** decisions the times also correlate substantially with the corresponding frequencies (r = .80).

In most cases where individuals had to "do something", they reported covert recitation from a specificable letter (90% for following decisions, and 95% for preceding). Figure 23 shows the frequencies with which various letters of the alphabet were reported as the beginning point on those trials when they engaged in covert recitation of a specific portion of the alphabet (Category B responses). This plot provides clear evidence that there are preferred points of entry into the alphabet, and that entry points are, to a considerable extent, shared by individuals. The deviation of this plot from a rectangular distribution (which would denote no preferred entry points) is clearly greater in early portions of the alphabet than in later portions. This could result from a lesser

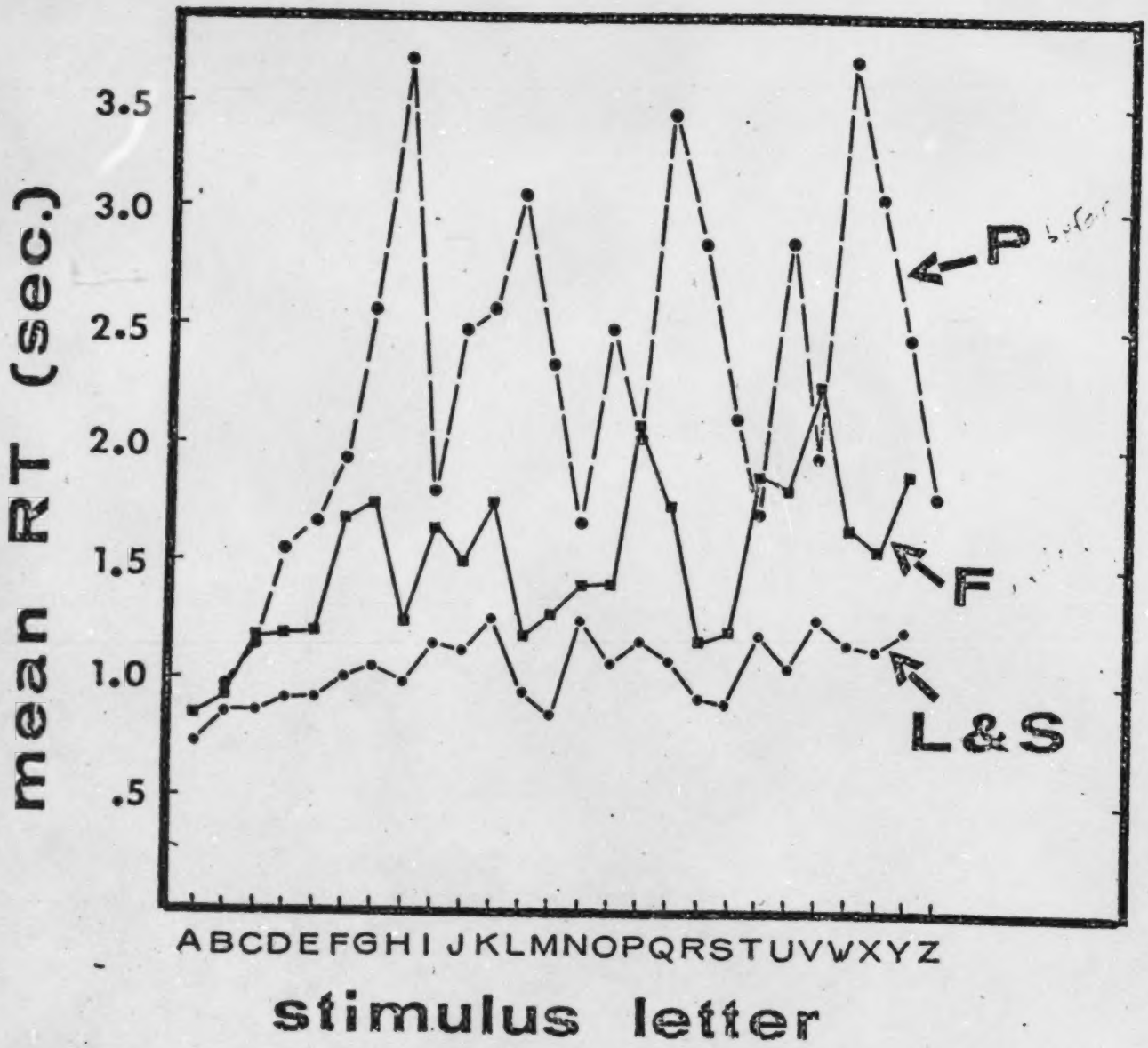


Figure 22: Mean RTs for adults (Exp. 2).

tendency for there to be stable preferred entry points later in the alphabet, or it might simply reflect the fact that there is less inter-individual consistency as to the location of those preferred points later in the alphabet than near the beginning. If the alphabet is viewed as a highly overlearned serial list, the regular peaks seem tc establish the validity of conceptions of serial list learning as the acquisition of a set of subjective subsequences or chunks, coupled with an order of the chunks.

We will use the maximum values of this distribution to provide an empirically based segmentation of the alphabet. Based on the peaks of the **before** curve in Figure 23, the alphabet appears to be segmented into chunks starting with the following letters: A, H, L, Q, U, X.

6.6 Description & Evaluation of ALPHA: a model of alphabetic access

In order to generate RT predictions from the model to compare with subjects' performance, we need to be more specific about its component processes and about how each process contributes to the overall RT. In this section we will describe a computer simulation model, **ALPHA**, for the **before** and **after** tasks used in this study. First, we will describe the data structure for the representation of the alphabet. Then we will discuss the processes that operate on this structure, and the number of parameters that could be associated with these processes. Finally, we will present the results of attempting to approximate all of the parameters with a two-parameter model for the **after** and **before** tasks.

6.6.1 Representation

In Figure 19a, each letter of the alphabet has a link pointing to the name of the chunk in which it can be found. The chunks are linked to their predecessor and successor chunk names, as well as to the actual list of alphabetic elements that comprise the chunk. These lists are accessable only through their beginnings, and only forward search is possible, since only the 'next' of each element is available in the representation.

The most important feature of this reperesentation is that prcbe letters do not have direct access to their *nexts* or *priors*; instead they have direct access only to the *name* of the chunk in which the probe is located. While this may at first seem non-intuitive, it is simply a formalization of the notion of preferred entry points. When people use these entry points, they do not choose among them at random. Rather, they tend to choose the one that is "just ahead" of the probe letter.

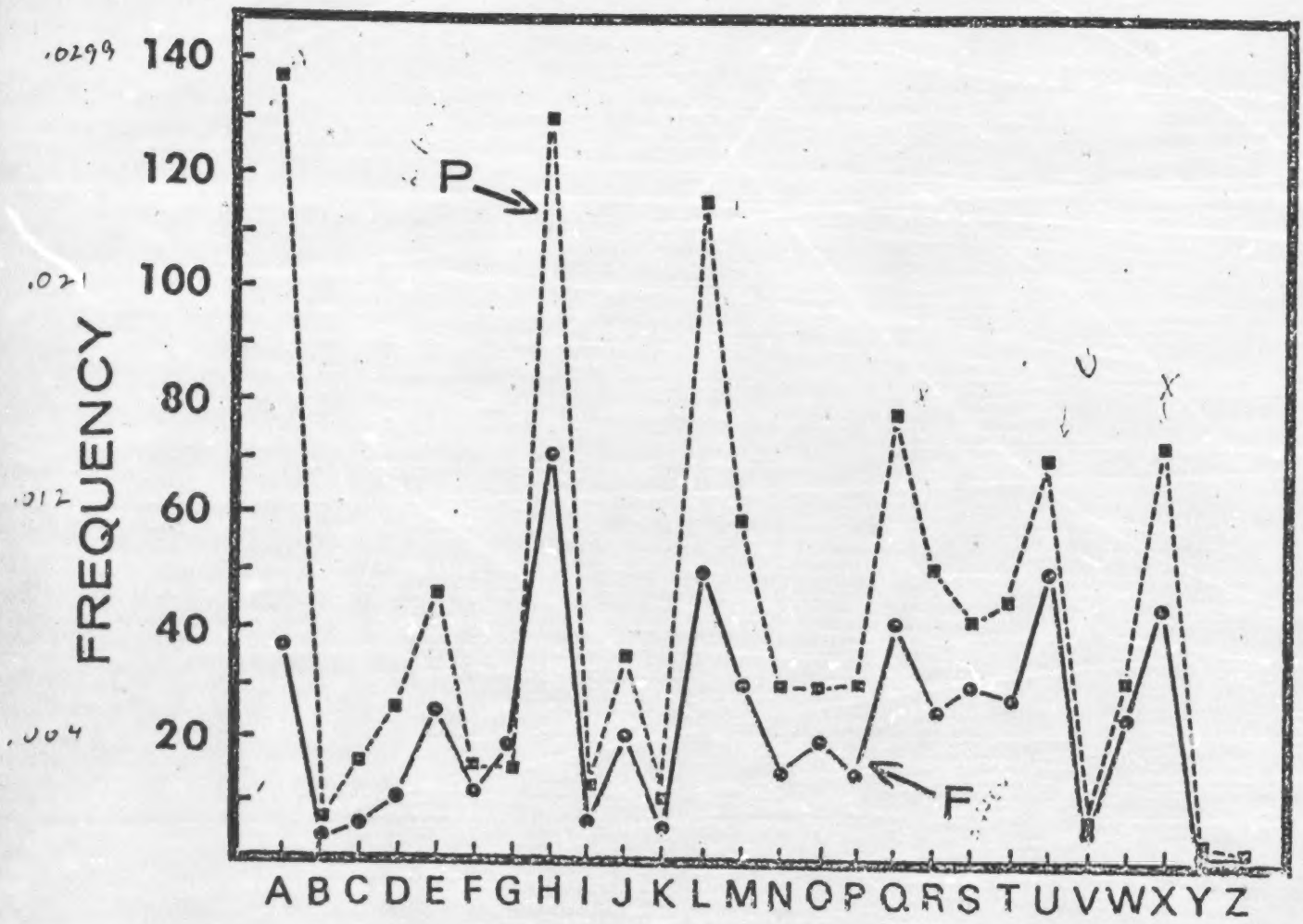


Figure 23: Frequency of reported entry points.

6.6.2 Processes

Figures 19b and 19c show the flow chart for ALPHA on the two tasks.[2] The flow chart shows the basic steps of the program, with some of the detail supressed for clarity of exposition. Associated with each of the processes that contribute to the differential times is a parameter: n_1 is the time to access the next chunk name; t_1 is the time to compare the chunk name with the chunk name of the probe; n_2 is the time to access the next item *within* a chunk, etc. The primed parameters correspond to similar processes for the **before** task.

6.6.3 Parameter estimation and model evaluation

The predicted RTs from ALPHA depend on both the assumed segmentation of the alphabet and the estimated values for all the parameters. Our goal in fitting the model to the data was to maintain our basic assumptions about process and structure, while limiting the number of degrees of freedom in the parameter estimation procedure. Therefore, we based the alphabetic segmentation not on the same data set used for the parameter estimates (the RTs), but rather on the peaks of the entry point frequency function described above (see Fig. 23). The parameter estimation procedure has been reduced to a linear regression involving the number of executions of a single internal process: doing a "next" on all of the internal list structures in ALPHA that are required to produce the response to a probe.[3] That is, for each of the 25 possible probes on the **after** or **before** task, the model, in addition to producing the answer, computes the number of times that it had to do a "next" on any of its internal structures. In this manner, an effort estimate is computed for each of the probes, and this effort estimate is regressed against the RTs. The regression produces an estimate of the basic "next" time, and this estimate can then be used to generate a predicted time for each probe.

Table 3 lists the number of "nexts" executed for each probe letter on the **after** and **before** tasks. The table also shows the mean RTs from experiment II, and the predicted RTs. The predictions are generated by substituting the number of "nexts" associated with each probe position into the regression equations shown in Table 4. The estimate of the time to do a "next" on a list is 127 msec for **after** and 153 msec for **before**, with the regression accounting for at least 50% of the variance. (If we use a segmentation based on the peaks of the RT curves, instead of the independent reported entry points, then the amount of variance accounted for increases to around 55%.) For example, Table 4 indicates that in response to **after** G, ALPHA requires 10 "nexts". Substituting n = 10 into

---

[2]The program is written in MACLISP, a variant of LISP used at Carnegie-Mellon Univeristy. Listings of the program and sample runs are available from the first author. People with access to the ARPA network can contact KLAHR@CMUA.

[3]This is a simple count of all CDRs used by all the functions in the LISP program as it does the **after** or **before** task.

the regression equation yields a predicted time of approximately 1870 msec, as shown in Table 4. The actual mean RT for **after** G is 1730 msec.

chunk	position	probe	AFTER nexts	AFTER RT	AFTER predicted	BEFORE nexts	BEFORE RT	BEFORE predicted
	1	A	3.0	85.00	99.77	—	—	—
	2	B	4.0	93.00	111.45	2.0	96.0	156.2
1	3	C	5.0	116.00	124.13	3.0	113.0	171.5
	4	D	6.0	118.00	136.80	4.0	155.0	186.8
	5	E	7.0	119.00	149.48	5.0	166.0	202.0
	6	F	8.0	169.00	162.16	6.0	193.0	217.3
	7	G	10.0	173.00	187.51	7.0	256.0	232.6
	8	H	4.0	124.00	111.45	16.0	377.0	370.1
2	9	I	5.0	164.00	124.13	3.0	179.0	171.5
	10	J	6.0	148.00	136.80	4.0	249.0	186.8
	11	K	8.0	173.00	162.16	5.0	256.0	202.0
	12	L	5.0	118.00	124.13	11.0	306.0	293.7
	13	M	6.0	127.00	136.80	4.0	233.0	186.8
3	14	N	7.0	142.00	149.48	5.0	166.0	202.0
	15	O	8.0	142.00	162.16	6.0	249.0	217.3
	16	P	10.0	209.00	187.51	7.0	202.0	232.6
	17	Q	6.0	174.00	136.80	14.0	340.0	339.5
4	18	R	7.0	115.00	149.48	5.0	273.0	202.0
	19	S	8.0	122.00	162.16	6.0	213.0	217.3
	20	T	10.0	186.00	187.51	7.0	171.0	232.6
	21	U	7.0	182.00	149.48	13.0	288.0	324.2
5	22	V	8.0	226.00	162.16	6.0	195.0	217.3
	23	W	10.0	167.00	187.51	7.0	378.0	232.6
	24	X	8.0	156.00	162.16	12.0	306.0	309.0
6	25	Y	9.0	189.00	174.83	7.0	247.0	232.6
	26	Z	—	—	—	8.0	177.0	247.9

x10 = sec

Table 3: The number of "nexts" executed for each probe letter on the **after** and **before** tasks.

Figure 24 contains a plot of predicted vs actual RTs (from Table 3) for **after** and **before** tasks. The most prominent feature of the RT curves are the local extreme points caused by chunk boundary crossings. ALPHA appears to be able to capture these quite well on both **after** and **before** tasks. On the **after** curves, the chunk boundaries at G-H and K-L show close correspondence between actual and predicted. The next two boundaries, P-Q, and T-U, have the appropriate maxima, but **after** R, although fast relative to **after** Q, is not a local min. Finally, there is an unpredicted local max at **after** V. The **before** curves are also in closer correspondence at the earlier chunk boundaries. The first four local maxima are exactly as predicted. Neither **before** M nor **before** R are the local mins they should be, while **before** I and **before** V are.

		R^2
after	RT = 607 + 127n	50%
before	RT = 1260 + 153n	54%

n = number of "nexts" on all lists and sublists

[Segmentation based on entry point Frequencies]

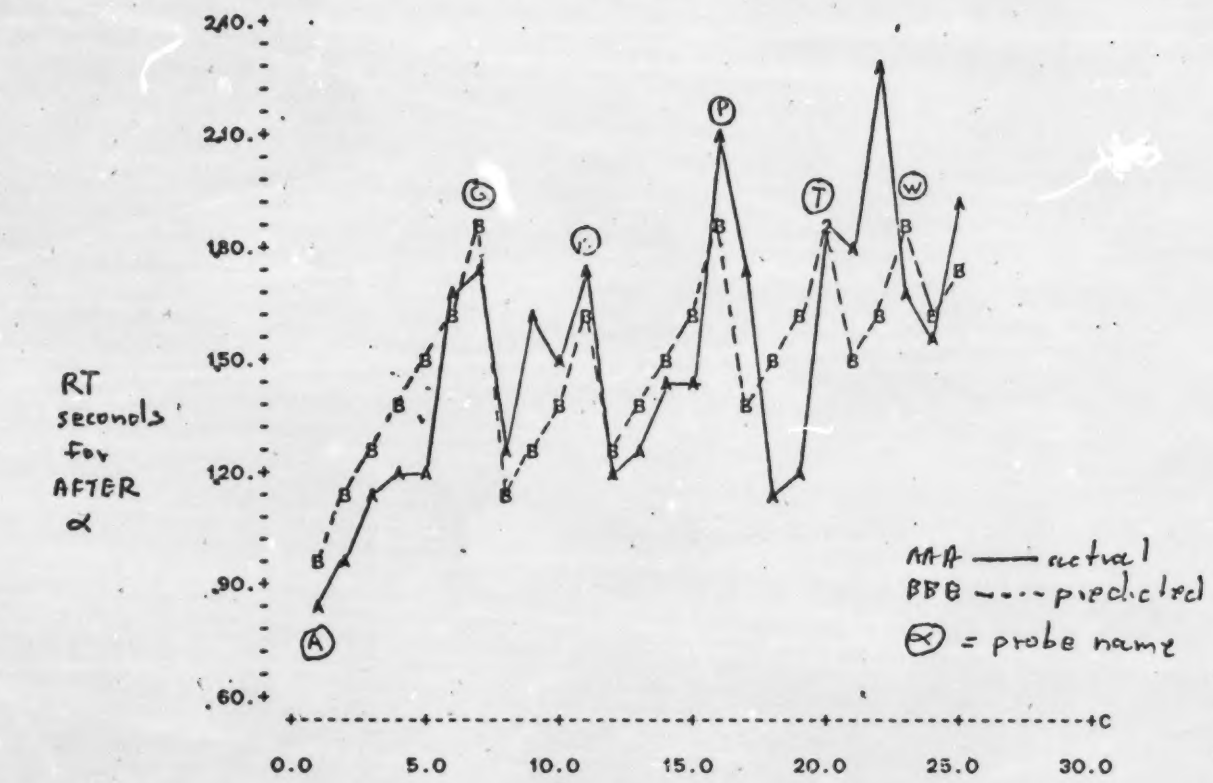
Table 4: Regression results for ALPHA effort against Exp. 2 data.

Perhaps the most interesting deviation from ALPHA's predictions are the slight but consistent over predictions for the first chunk, particularly for the **before** times for the first few letters. This might result from alternative representations for these items ("the ABC's") that provide more direct access than the full process modeled by ALPHA.

6.7 Children's Alphabetic Access

Essentially the same procedure was used to study children's processing of the alphabet. Eight 6-year old children were presented with the **before** and **after** tasks as described in section 6-4; the major difference was that each letter was presented only three times. Their RT patterns showed the same general segmentation of the alphabet, and the same effect of chunk boundaries. Regression results indicated that ALPHA could account for about 35% of the RT variance. However, the most interesting difference was that the *rate* of processing for children was around 500 msec per next, even though the same model seems to apply for adults and children.

Thus, in two quite distinct domains, we have found examples of identical structures and processes in children and adults, and yet we still find large differences in processing rates for elementary steps.



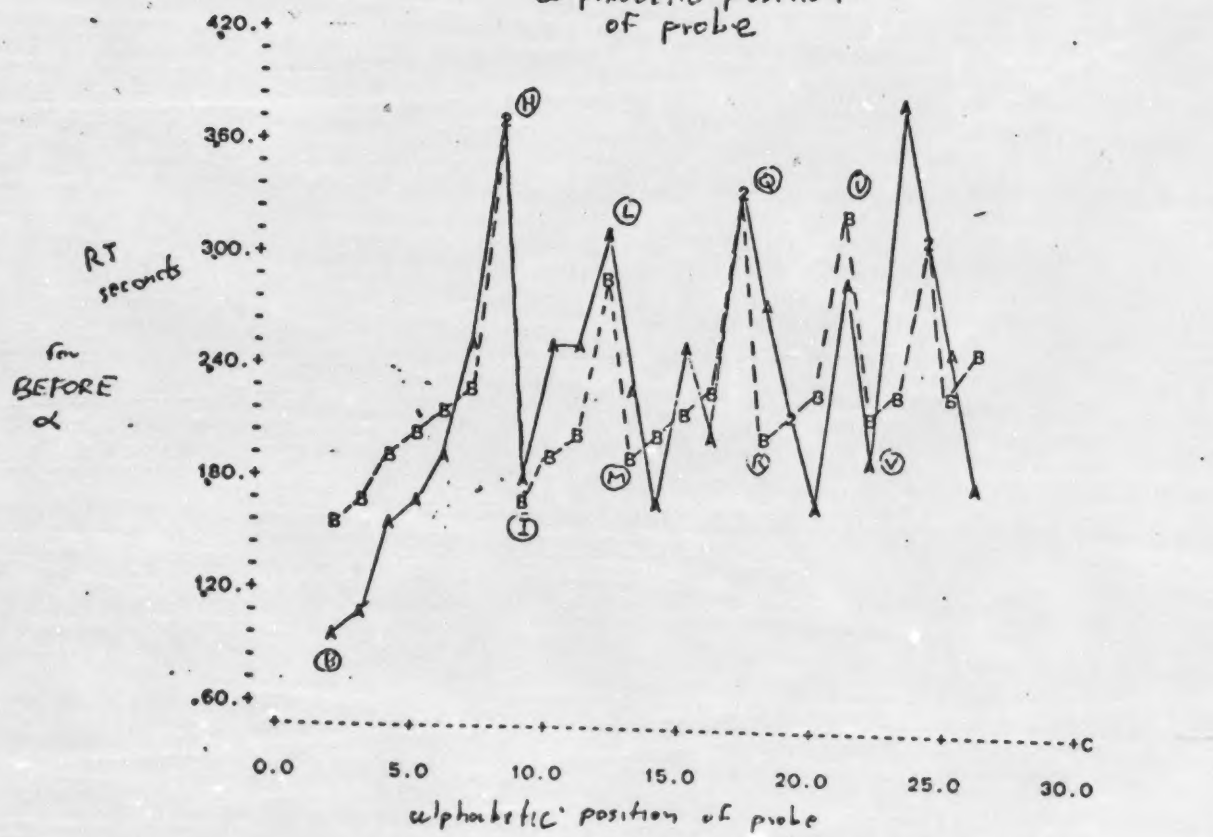


Figure 24: Prediction and actual RTs from "tuned" ALPHA.

7 Instructional Theory

With respect to instructional theory, in collaboration with Robert Siegler, I have been investigating theoretical and empirical approaches to the question of how the acquisition of new knowledge in children is related to the prior existence of old knowledge (Siegler & Klahr, 1981). We address issues of critical stages, instructional readiness and the fit between a child's current level of knowledge and the kind of instructional material with which he is presented. We draw on examples from a wide range of task domains, including classic Piagetian concepts about balance scales, time. speed, distance, proportionality, and so on, as well as on the work mentioned above in problem solving and planning.

The chapter opens with a broad question: "When do children learn?" All of the research reviewed in the chapter points to a single general answer: "Children learn when there is an appropriate relationship between their existing knowledge and the instructional material presented to them." This formulation indicates that the important tasks are to determine children's existing knowledge, the relationship between their existing knowledge and alternative instructional material that might be presented to them, and the ideal relationship between existing knowledge and instructional material. We describe several distinct examples of rule-assessment research aimed at addressing these issues. There are four conclusions towards which this evidence seems to converge:

1. Children's knowledge can be characterized in terms of rules. This conclusion seems to be equally valid for simple concepts, for more complex concepts, for procedural knowledge in which sets of rules are combined into strategies, and for the types of rules for learning that are embodied in self-modifying production systems. The contents of all of these types of rules can be determined by presenting problems on which different rules lead to distinct patterns of responses.

2. In cases in which children use two or more partially correct rules before mastering a concept or procedure, the partially correct rules are ordered in terms of increasing correlation with the predictions of the mastery rule. While there may be declines in the proportion of correct responses in limited subsets of the problem domain (as in conflict-weight problems on the balance scale), children will adopt only those rules that lead to an overall increase in the proportion of correct responses.

3. The effectiveness of a learning experience is in part determined by whether the learning experience discriminates the child's existing rule from the correct rule. On the time concept, younger children benefited from problems that discriminated end points from time, but those same problems had no effect on older children. Older children benefited from problems that discriminated distance from time, but those same problems had no effect on younger ones. In each case, this was due to the

relationship between what the child already knew and what dimensions were discriminated by the training problems. The similarity of the predictions generated by many of the Tower of Hanoi rules for most of the possible problems suggests that the same phenomenon might well emerge there. Without assessments of a child's existing knowledge, it would be extremely difficult, if not impossible, to anticipate which of the Tower of Hanoi or time concept problems would provide useful learning experiences for the child and which would not. With such assessments, there is a principled way of predicting.

4. A major reason why children do not immediately adopt the correct rule for all concepts is their limited encoding of the correct rule's component dimensions. Children's encoding of a dimension may be inadequate due to lack of knowledge of the dimension's importance, lack of perceptual salience of the dimension in the situation in which the concept is to be applied, or lack of adequate ability to hold all of the relevant information in memory.

8 Summary

Pre-school children's problem-solving processes have been investigated during the grant period in both direct and indirect ways. The direct investigations have focused on substantive and methodological issues related to how children solve a few well defined puzzles. The indirect work has dealt with related issues: non-monotone developmental curves, rates of processing, structure-process invariance and instructional theory.

An important substantive contribution of this research is the discovery that by the time they reach Kindergarten, children appear to have acquired many of the components of mature problem solving strategies. These components are acquired without direct instruction, and there is substantial variation in the particular components that exist in different children's repertoires. Therefore, any attempt to instruct children to be better problem solvers must first make a careful determination not only of the level of their performances, but, more importantly, precisely what strategies they are using. Another contribution of this research has been the development of a methodology to facilitate such a determination.

The focus on fine-grained characterization of underlying processes has also enabled us to propose an interpretation of non-monotone growth curves, and we have argued that these surface measures do not reflect any interesting underlying processes. Finally, our focus on rates, processes and structures as potential sorces of developmental differences, has provided a potentially fruitful area for further investigations of how children learn to solve problems.

9 Papers Published During Grant Period

Klahr, D. Goal Formation, Planning, and Learning by Pre-school Problem Solvers, or: 'My socks are in the dryer'. In R.S. Siegler(Ed.), Children's Thinking: What Develops?, Lawrence Erlbaum Associates, Hillsdale, NJ, pages 181-212, 1978.

Klahr, D. Toward an information processing theory of cognitive development. In R. Klewe & H. Spada (Eds.), Developmental Models of Thinking. New York: Academic Press, 1980. [German Edition: Studien zur Denkentwicklung. Bern, Stuttgart, Wien: Huber, 1981.]

Klahr, D. Non-monotone assessment of monotone development: An information processing analysis. In S. Strauss & R. Stavy (Eds.), U-Shaped behavioral growth, New York: Academic Press, 1981.

Siegler, R.S., & Klahr, D. When do children learn? The relationship between existing knowledge and the acquisition of new knowledge. In R. Glaser (Ed.), Advances in Instructional Psychology, Vol.2, Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Klahr, D., & Robinson, M. Formal Assessment of problem-solving and planning proceesses in pre-school children. Cognitive Psychology, 13, 1981, 113-148.

10 Professional Activities During Grant Period

March 1978	Paper presentation at Annual Meeting of AERA, Toronto.
April 1978	Invited paper at IEEE Computer Society Workshop on Pattern Recognition and Artificial Intelligence. Princeton, NJ.
August 1978	Panel Chairman at NIE Research Conference on Te ing, Falmouth, Massachusetts.
Oct. 1978	Colloquium: National Institute of Education, Washington, DC.
Nov. 1978	Colloquium: Children's Problem Solving, Department of Educational Psychology, McGill University, Montreal, Canada.
Nov. 1978	Paper presentation: Psychonomic Society Meeting, San Antonio, TX.
Dec. 1978	Visiting Scholar: Department of Psychology, University of Iowa, (1 week).
March 1979	Paper presentations at the biennial meeting of the Society for Research in Child Development, San Francisco.
April 1979	Invited symposium, annual meetings of AERA, San Francisco.
Oct. 1979	Colloquium: Department of Psychology, Stanford University.
Nov. 1979	Paper presention, Psychonomic Society Meeting, Phoenix, Arizona.
Nov. 1979	Invited participant: Wingspread Conference on Basic Processing in Mathematics Learning, Racine, Wisconsin.
Feb., 1980	Visiting Scholar, School of Education, Deakin University, Geelong, Victoria, Australia, (3 weeks),
April, 1980	Colloquium: School of Education, Stanford University.
April, 1980	Colloquium: Group in Science and Mathematics Education, University of California, Berkeley.
March, 1981	Colloquium: Graduate Center, City University of New York.
April, 1981	Symposium: Biennial Meeting, SRCD. Boston, Mass.

References

Anderson, J.R., Kline, P.J., & Beasley, C.M., Jr. *A general learning theory and its application to schema abstraction.* Technical Report 78-2, Department of Psychology, Carnegie-Mellon University, 1978.

Anzai, Y., & Simon, H.A. The theory of learning by doing. *Psychological Review*, 1979, *86*, 124-140.

Anzai, Y. Learning strategies by computer. In *The Proceedings of the 2nd Conference on Computational Studies of Intelligence,* Toronto: Canadian Society for Computation Studies of Intelligence, 1978.

Byrnes, M.M. & Spitz, H.H. Developmental progression of preformance on the Tower of Hanoi Problem. *Bulletin of the Psychonomic Society*, 1979, *14*, 379-381.

Chi, M. T. H., & Klahr, D. Span and rate of apprehension in children and adults. *Journal of Experimental Child Psychology*, 1975, *19*, 434-439.

Elffers, J. *Tangram: The Ancient Chinese Shapes Game.* New York: Penguin Books 1976.

Forgy, C.L. *OPS4 User's Manual.*. Technical Report, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, July, 1979.

Karmiloff-Smith, A. Micro- and macro-developmental changes in language acquisition and other representational systems. *Cognition*, 1979, *3*, 91-118.

Klahr, D., & Wallace, J. G. The development of serial completion strategies: An information processing analysis. *British Journal of Psychology*, 1970, *61*, 243-257.

Klahr, D., & Wallace, J. G. An information processing analysis of some Piagetian experimental tasks. *Cognitive Psychology*, 1970, *1*, 358-387.

Klahr, D., & Wallace, J.G. Reply to Hayes: On the value of theoretical precision. In S. Farnham-Diggory (Ed.), *Information processing in children,* New York: Academic Press, 1972.

Klahr, D., & Wallace, J. G. The role of quantification operators in the development of conservation of quantity. *Cognitive Psychology*, 1973, *4*, 301-327.

Klahr, D., & Wallace, J.G. *Cognitive development: An information processing view.* Hillsdale, N.J.: Lawrence Erlbaum Associates 1976.

Klahr, D. An information processing approach to the study of cognitive development. In A. Pick (Ed.), *Minnesota symposia on child psychology, Vol. 7,* Minneapolis: University of Minnesota Press, 1973.

Klahr, D. A production system for counting, subitizing and adding. In W. G. Chase (Ed.), *Visual information processing,* New York: Academic Press, 1973.

Klahr, D. Quantification processes. In W. G. Chase (Ed.), *Visual information processing,* New York: Academic Press, 1973.

Klahr, D. Designing a learner: Some questions. In D. Klahr (Ed.), *Cognition and instruction,* Hillsdale, N.J.: Lawrence Erlbaum Associates, 1976.

Klahr, D. Steps toward the simulation of intellectual development. In L. B. Resnick (Ed.), *The nature of intelligence*, Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1976.

Klahr, D. Goal formation, planning, and learning by pre-school problem solvers, or: 'My socks are in the dryer'. In R.S. Siegler (Ed.), *Children's thinking: What develops?*, Hillsdale, N.J.: Lawrence Erlbaum Associates, 1978.

Klahr, D. Non-monotone assessment of monotone development: An information processing analysis. In S. Strauss & R. Stavy (Eds.), *U-shaped behavioral growth.*, New York: Academic Press, 1981.

Langley, P. Finding Common Paths as a Learning Mechanism. Unpublished manuscript, Carnegie-Mellon University.

Lewis, C.H. *Production system models of practice effects*. PhD thesis, University of Michigan, , 1978.

Lovelace, E.A., & Spence, W.A. Reaction Times for Naming Successive Letters of the Alphabet. *Journal of Experimental Psychology*, 1972, *94*, 231-233.

Lovelace, E.A., Powell, C.M., & Brooks, R.J. Alphabetic Position Effects in Covert and Overt Alphabetic Recitation Times. *Journal of Experimental Psychology*, 1973, *99*, 405-408.

Luchins, A.S. Mechanization in problem solving--The effects of 'Einstellung'. *Psychological Monographs*, 1942, *54(248)*, .

Piaget, J. *The grasp of consciousness.* Cambridge, Mass.: Harvard University Press 1976.

Siegler, R.S., and Klahr, D. When do children learn? The relationship between existing knowledge and the acquisition of new knowledge. In R. Glaser (Ed.), *Advances in Instructional Psychology, Vol.2*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Strauss, S., & Stavy, R. U-shaped behavioral growth: Implications for theories of development. In W.W. Hartup (Ed.), *Review of Child Development Research, Vol. 6*, Chicago: University of Chicago Press, 1980.

Appendices

- Appendix A is an annotated listing of the MACLISP programs for the nine models. Pages A-9 and A-10 show some examples of how to use the models.

- Appendix B is a listing of the production system version of Model 9, written in OPS4. The trace on B-3 shows the model solving a seven move flat to flat problem. If followed carefully, it should provide a good feel for the operation of the production system and the complexity of the model.

```
;;;TQH5 models, second pass
;;;state is unordered list of peg-contents lists.
;;;peg-contents list is (pegx Cn CN Cn) or (pegx)( see SETUP)
;;;MODs return move-newstate pairs (or just move)
;     or (nil (DONE)). e.g. ( (c2 pega)((pega c2 c1)(pegb c3)(pegc)))
;;;moves non-destructive: return revised states
;;;most functions need state variable as input
;;;Global vars:  CANORDER (see SETUP),TRACEON,PROBSTATES
;;;
;;;to run, first do (setup), then (solve 'FN-NAME initial goal),
;;;       using initial and goal states assigned by SETUP.
;;;     e.g (solve 'mod10 towa flat2)
;;;
;;;NOTE: Only MOD10 is guarenteed not to loop. However SOLVE
;;;      should provide the right environment for testing any
;;;      new functions.
;;;to get only next move, do (NEXTMOVE 'fn-name initial goal)
;;;
;;; to get problem N from PROBSTATES do (nthprob N), then
;;;   set values to its CAR and CADR for inital and final states
;;;
;;;SETUP creates a few initial and goal states
;;;     and calls setup2.
;;;SETUP2 Uses SLURP to load function
;;;     which does actual setup of PROBSTATES (full problem set).
;;;TRACEOFF/TRACEON control output from TRY10 and LEGAL

(COMMENT --CONTENTS-- ALLMOVES ALL_1MOVES ALL_2MOVES ALL_3MOVES ANY1S
        ANY2S ANY3S ANYNS AVOIDDOUBS BEFORE CULP10 DIFFS EMPTY ISIN
        LEGAL LEGAL_MOVES MAKE MINDIFF MOD1 MOD10 MOD2 MOD3 MOD34
        MOD4 MOD5 MOD6 MOD7 MOD8 MOD8A MOD8B MOD9 MOVE NEXTMOVE
        NEXTSTATES NTHPROB OKORDER OTHER PEGOF PICKUP PLACE REMAINS
        REMOVE SEEPEG SETUP SETUP2 SOLVE SOLVED STRIPP TOPCAN
        TRACEOFF TRACEON TRY10 TRY2 TRY3 TRY4 MODPROB)
```

FILMED FROM
BEST COPY AVAILABLE

A-2

```
;;;generate  table of first moves for all modles included in second list
;;;first list is used to print corresponding headings
;;;output to TTY, so use DRIBBLE to save file

(DEFUN ALLMOVES NIL
  (PROG (ALLPROBS PROBSTATE INIT FIN CP)
        (PRINT 'PB)
        (MAPC '(LAMBDA (N) (PRINC '|   |) (PRIN1 'MOD) (PRIN1 N))
              (LIST 1. 2. 3. 4. 5. 6. 7. 8. 9.))
        (SETQ ALLPROBS PROBSTATES)
   LOOP (SETQ PROBSTATE (CAR ALLPROBS))
        (SETQ INIT (CAADR PROBSTATE))
        (SETQ FIN (CADADR PROBSTATE))
        (PRINT (CAR PROBSTATE))
        (MAPC '(LAMBDA (FN)
                 (PRINC '|  |)
                 (SETQ CP (NEXTMOVE FN INIT FIN))
                 (PRIN1 (CAR CP))
                 (PRINC '_)
                 (PRIN1 (CAADR CP))
                 (PRIN1 (CADADR CP)))
              (LIST 'MOD1
                    'MOD2
                    'MOD3
                    'MOD4
                    'MOD5
                    'MOD6
                    'MOD7
                    'MOD8B
                    'MOD10))
        (SETQ ALLPROBS (CDR ALLPROBS))
        (COND ((NULL ALLPROBS) (TERPRI) (RETURN 'DONE))
              (T (GO LOOP)))))

;;;ALL-1MOVES returns all legal single moves from S
;;;format: ( ( (can peg)(new state))((can peg)(new state))..etc)
;;;
;;;ALL2_MOVES returns all legal double moves from STATE, except
;;;  double moves of same can
;;;format:( (<move1><move2><newstate>)(<move1><move2><newstate>)..etc)
;;;
;;;ALL_3MOVES  similar to above..but three move look-ahead
;;;
;;;

(DEFUN ALL_1MOVES (S)
  (COMMENT RETURN LISTS OF ALL SINGLE MOVES AND STATES FROM STATE)
  (PROG (MV MVSTATES NEXTMOVES)
        (SETQ NEXTMOVES (LEGAL_MOVES S))
   LOOP (SETQ MV (CAR NEXTMOVES))
        (SETQ MVSTATES
              (CONS (LIST MV (MOVE (CAR MV) (CADR MV) S)) MVSTATES))
        (SETQ NEXTMOVES (CDR NEXTMOVES))
        (COND ((NULL NEXTMOVES) (RETURN MVSTATES))
              (T (GO LOOP)))))

(DEFUN ALL_2MOVES (STATE)
  (COMMENT ALL TWO CAN MOVES FROM CURRENT STATE)
  (PROG (MVSTATE MVSTATES LEGALS TWODEEP)
        (SETQ MVSTATES (ALL_1MOVES STATE))
   LOOP (SETQ MVSTATE (CAR MVSTATES))
        (SETQ LEGALS
              (AVOIDDOUBS (CAR MVSTATE)
                          (LEGAL_MOVES (CADR MVSTATE))))
        (SETQ TWODEEP
              (APPEND (MAPCAR '(LAMBDA (MV)
                                 (LIST (CAR MVSTATE)
                                       MV
                                       (MOVE (CAR MV)
                                             (CADR MV)
                                             (CADR MVSTATE))))
                              LEGALS)
                      TWODEEP))
        (SETQ MVSTATES (CDR MVSTATES))
        (COND ((NULL MVSTATES) (RETURN TWODEEP))
              (T (GO LOOP)))))

(DEFUN ALL_3MOVES (STATE)
  (COMMENT ALL THREE CAN MOVES FROM CURRENT STATE)
  (PROG (MVSTATE MVSTATES LEGALS THREEDEEP)
```

A3

```
        (SETQ MVSTATES (ALL_2MOVES STATE))
   LOOP (SETQ MVSTATE (CAR MVSTATES))
        (SETQ LEGALS
              (AVOIDDOUBS (CADR MVSTATE)
                          (LEGAL_MOVES (CADDR MVSTATE))))
        (SETQ THREEDEEP
              (APPEND (MAPCAR '(LAMBDA (MV)
                                 (LIST (CAR MVSTATE)
                                       (CADR MVSTATE)
                                       MV
                                       (MOVE (CAR MV)
                                             (CADR MV)
                                             (CADDR MVSTATE))))
                              LEGALS)
                      THREEDEEP))
        (SETQ MVSTATES (CDR MVSTATES))
        (COND ((NULL MVSTATES) (RETURN THREEDEEP))
              (T (GO LOOP)))))

;;;
;;;ANY1S, ANY2S, ANY3S, all call ANYNS to search lists of n-move final states
;;;MVSTATES for any satisfied GOALS (GOALS is in same format as DIFFS)
;;;e.g., does (CAN2 PEGB) exist in one of the final states generated by
;;;ALL_3MOVES?
;;;

(DEFUN ANY1S (GOALS MVSTATES) (ANYNS 1. GOALS MVSTATES))

(DEFUN ANY2S (GOALS MVSTATES) (ANYNS 2. GOALS MVSTATES))

(DEFUN ANY3S (GOALS MVSTATES) (ANYNS 3. GOALS MVSTATES))

(DEFUN ANYNS (N GOALS MVSTATES)
  (PROG (GS MVS)
        (SETQ GS GOALS)
   GLOOP (COND ((NULL GS) (RETURN NIL)))
        (SETQ MVS MVSTATES)
   STLOOP (COND ((NULL MVS) (SETQ GS (CDR GS)) (GO GLOOP)))
        (COND ((ISIN N (CAR GS) (CAR MVS)) (RETURN (CAAR MVS))))
        (SETQ MVS (CDR MVS))
        (GO STLOOP)))

(DEFUN AVOIDDOUBS (MV MVLIST)
  (COMMENT DELTE ANY OCCURRENCES OF CAN IN MV FROM MVLIST)
  (COND ((NULL MVLIST) NIL)
        ((EQ (CAR MV) (CAAR MVLIST)) (AVOIDDOUBS MV (CDR MVLIST)))
        (T (CONS (CAR MVLIST) (AVOIDDOUBS MV (CDR MVLIST))))))

(DEFUN BEFORE (X Y L)
  (COMMENT T IF X BEFORE Y IN L ,NIL OTHERWISE)
  (> (LENGTH (MEMBER X L)) (LENGTH (MEMBER Y L))))

;;;CULP10 is "perceptual part of MOD10. Detects culprits when move
;;;      is know to be illegal.
;;;

(DEFUN CULP10 (CAN TO STATE)
  (COMMENT RETURN SMALLEST OBSTRUCTOR TO MOVE)
  (COMMENT SPECIF TO 3-CAN CONGIFURATIONS)
  (COND ((EQ CAN 'C2) 'C3)
        ((OR (MEMBER 'C2 (SEEPEG (PEGOF CAN STATE) STATE))
             (MEMBER 'C2 (SEEPEG TO STATE)))
         'C2)
        (T 'C3)))

;;;
;;;DIFFS uses CANORDER to guarantee min can first
;;;

(DEFUN DIFFS (S1 S2)
  (COMMENT RETURN LIST OF SUBGOALS AS CAN-GOALPEG PAIRS)
  (REVERSE (REMOVE NIL
                   (MAPCAR '(LAMBDA (CAN)
                              (COND ((NOT (EQ (PEGOF CAN S1)
                                              (PEGOF CAN S2)))
                                     (LIST CAN (PEGOF CAN S2)))
                                    (T NIL)))
                           CANORDER))))

(DEFUN EMPTY (STATE)
  (COMMENT RETURN FIRST EMPTY PEG OR NIL)
```

A-4

```
  (COND ((NULL STATE) NIL)
        ((NULL (CDAR STATE)) (CAAR STATE))
        (T (EMPTY (CDR STATE)))))

(DEFUN ISIN (N G MVST)
  (SETQ STATE (NTH N MVST))
  (COND ((MEMBER (CAR G) (ASSOC (CADR G) STATE)) T)
        (T NIL)))

(DEFUN LEGAL (CAN TO STATE)
  (COND ((NOT (EQ CAN (TOPCAN (PEGOF CAN STATE) STATE)))
         (COND (TRACEON (PRINT '|can not free to move|)))
         NIL)
        ((NOT (OKORDER CAN (TOPCAN TO STATE)))
         (COND (TRACEON (PRINT '|destination peg blocked|)))
         NIL)
        (T T)))

(DEFUN LEGAL_MOVES (STATE)
  (COMMENT RETRUN ALL LEGAL MOVES FROM STATE)
  (PROG (MVLST MOVES)
        (SETQ MVLST MOVELIST)
   LOOP (COND ((LEGAL (CAAR MVLST) (CADAR MVLST) STATE)
               (SETQ MOVES (CONS (CAR MVLST) MOVES))))
        (SETQ MVLST (CDR MVLST))
        (COND ((NULL MVLST) (RETURN MOVES))
              (T (GO LOOP)))))

(DEFUN MAKE (MOVE_PEG STATE)
  (LIST MOVE_PEG (MOVE (CAR MOVE_PEG) (CADR MOVE_PEG) STATE)))

;;;
;;;MINDIFF expects subgoal order from DIFFS, so min is just CAR
;;;

(DEFUN MINDIFF (X Y) (CAR (DIFFS X Y)))

;;;Models from Klahr & RObinson paper
;;;In revised paper, Model 8 is MOD8B here, Model 9 is MOD10 here,
;;;     and MOD8 is not used.
;;;

(DEFUN MOD1 (C G)
  (COMMENT RETURN MODEL1 MOVE AND NEW STATE ,PROBABLY ILLEGAL)
  (MAKE (MINDIFF C G) C))

(DEFUN MOD10 (CURRENT GOAL)
  (COMMENT TRY10 ON MINIMUM CAN NOT ON GOAL PEG)
  (COMMENT RETURN LIST OF PAIRS OF LISTS MOVE-NEWSTATE OR NIL-DONE)
  (COND ((NULL (DIFFS CURRENT GOAL)) '(NIL (DONE)))
        (T (TRY10 (CAR (MINDIFF CURRENT GOAL))
                  (CADR (MINDIFF CURRENT GOAL))
                  CURRENT))))

(DEFUN MOD2 (CURRENT GOAL)
  (COND ((NULL (DIFFS CURRENT GOAL)) '(NIL (DONE)))
        (T (TRY2 (CAR (MINDIFF CURRENT GOAL))
                 (CADR (MINDIFF CURRENT GOAL))
                 CURRENT))))

;;;MOD3 and MOD4 differ only in use of TRY3 or TRY4, (i.e. focus on
;;;     to peg or from peg.
;;;
;;;

(DEFUN MOD3 (CURRENT GOAL) (MOD34 'TRY3 CURRENT GOAL))

(DEFUN MOD34 (FN CURRENT GOAL)
  (PROG (XCAN XPEG NEWMOVE NEWSTATE NDIFF)
    (SETQ NDIFF (MINDIFF CURRENT GOAL))
    (COND ((NULL NDIFF) (RETURN '(NIL (DONE))))
          (T (SETQ NEWMOVE
                   (FUNCALL FN (CAR NDIFF) (CADR NDIFF) CURRENT))))
    (COND ((LEGAL (CAR NEWMOVE) (CADR NEWMOVE) CURRENT)
           (RETURN (MAKE NEWMOVE CURRENT)))
          (T (SETQ XCAN (CAR NEWMOVE))
             (SETQ XPEG
                   (OTHER (PEGOF (CAR NEWMOVE) CURRENT)
                          (CADR NEWMOVE)))
             (RETURN (MAKE (LIST XCAN XPEG) CURRENT))))))
```

```
(DEFUN MOD4 (CURRENT GOAL) (MOD34 'TRY4 CURRENT GOAL))

;;;MOD5 and MOD6 do breadth or depth first search, respectively.
;;;MOD6 uses similar structure to MOD5, but looks at 1 goal at each
;;;depth(via LOOP) before moving on to next subgoal.
;;;

(DEFUN MOD5 (CUR GOL)
  (PROG (NDIFFS CANPEG1 CANPEG2)
    (SETQ NDIFFS (DIFFS CUR GOL))
    (COND ((NULL NDIFFS) (RETURN '(NIL (DONE))))
          ((SETQ CANPEG1 (ANY1S NDIFFS (ALL_1MOVES CUR)))
           (RETURN (MAKE CANPEG1 CUR)))
          ((SETQ CANPEG2 (ANY2S NDIFFS (ALL_2MOVES CUR)))
           (RETURN (MAKE CANPEG2 CUR)))
          (T (RETURN NIL)))))

(DEFUN MOD6 (CUR GOL)
  (PROG (XDIFF CANPEG NDIFFS)
        (SETQ NDIFFS (DIFFS CUR GOL))
   LOOP (SETQ XDIFF (LIST (CAR NDIFFS)))
        (COND ((NULL XDIFF) (RETURN '(NIL (DONE))))
              ((SETQ CANPEG (ANY1S XDIFF (ALL_1MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              ((SETQ CANPEG (ANY2S XDIFF (ALL_2MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              (T (SETQ NDIFFS (CDR NDIFFS)) (GO LOOP)))))

(DEFUN MOD7 (CUR GOAL)
  (PROG (NDIFF TPEG FPEG 3PEG)
    (SETQ NDIFF (MINDIFF CUR GOAL))
    (COND ((NULL NDIFF) (RETURN '(NIL (DONE))))
          ((LEGAL (CAR NDIFF) (CADR NDIFF) CUR)
           (RETURN (MAKE NDIFF CUR)))
          (T (SETQ TPEG (CADR NDIFF))
             (SETQ FPEG (PEGOF (CAR NDIFF) CUR))
             (SETQ 3PEG (PEGOF 'C3 CUR))
             (RETURN (MAKE (LIST 'C3
                                 (COND ((OR (EQ 3PEG TPEG)
                                            (EQ 3PEG FPEG))
                                        (OTHER FPEG TPEG))
                                       (T (OTHER 3PEG FPEG))))
                           CUR))))))

;;;MOD8 and MOD8a not used. "True" mod8 is MOD8B.
;;;

(DEFUN MOD8 (CUR GOL)
  (PROG (NDIFF TPEG FPEG 3PEG SUBCAN SUBPEG)
    (SETQ NDIFF (MINDIFF CUR GOAL))
    (COND ((NULL NDIFF) (RETURN '(NIL (DONE))))
          ((LEGAL (CAR NDIFF) (CADR NDIFF) CUR)
           (RETURN (MAKE NDIFF CUR))))
    (SETQ SUBCAN (CULP10 (CAR NDIFF) (CADR NDIFF) CUR))
    (SETQ SUBPEG (OTHER (PEGOF (CAR NDIFF) CUR) (CADR NDIFF)))
    (COND ((LEGAL SUBCAN SUBPEG CUR)
           (RETURN (MAKE (LIST SUBCAN SUBPEG) CUR)))
          (T (SETQ TPEG (CADR NDIFF))
             (SETQ FPEG (PEGOF (CAR NDIFF) CUR))
             (SETQ 3PEG (PEGOF 'C3 CUR))
             (RETURN (MAKE (LIST 'C3
                                 (COND ((OR (EQ 3PEG TPEG)
                                            (EQ 3PEG FPEG))
                                        (OTHER FPEG TPEG))
                                       (T (OTHER 3PEG FPEG))))
                           CUR))))))

(DEFUN MOD8A (CUR GOL)
  (PROG (XDIFF CANPEG NDIFFS 3PEG TPEG FPEG MNDIFF)
        (SETQ NDIFFS (DIFFS CUR GOL))
        (SETQ MNDIFF (MINDIFF CUR GOAL))
   LOOP (SETQ XDIFF (LIST (CAR NDIFFS)))
        (COND ((NULL XDIFF) (RETURN '(NIL (DONE))))
              ((SETQ CANPEG (ANY1S XDIFF (ALL_1MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              ((SETQ CANPEG (ANY2S XDIFF (ALL_2MOVES CUR)))
               (RETURN (MAKE CANPEG CUR))))
        (SETQ NDIFFS (CDR NDIFFS))
        (COND ((EQ (CAAR NDIFFS) 'C3)
               (SETQ TPEG (CADR MNDIFF))
               (SETQ FPEG (PEGOF (CAR MNDIFF) CUR))
```

A-6

```
                (SETQ 3PEG (PEGOF 'C3 CUR))
                (RETURN (MAKE (LIST 'C3
                                    (COND ((OR (EQ 3PEG TPEG)
                                               (EQ 3PEG FPEG))
                                           (OTHER FPEG TPEG))
                                          (T (OTHER 3PEG FPEG))))
                              CUR)))
               (T (GO LOOP)))))

(DEFUN MOD8B (CUR GOL)
  (PROG (XDIFF CANPEG NDIFFS)
        (SETQ NDIFFS (DIFFS CUR GOL))
   LOOP (SETQ XDIFF (LIST (CAR NDIFFS)))
        (COND ((NULL XDIFF) (RETURN '(NIL (DONE))))
              ((SETQ CANPEG (ANY1S XDIFF (ALL_1MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              ((SETQ CANPEG (ANY2S XDIFF (ALL_2MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              ((SETQ CANPEG (ANY3S XDIFF (ALL_3MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              (T (SETQ NDIFFS (CDR NDIFFS)) (GO LOOP)))))

;;;MOD9 not used. But note kludge fro switching subgoal order and trying again
;;;

(DEFUN MOD9 (CUR GOL)
  (PROG (XDIFF CANPEG NDIFFS MV)
        (SETQ NDIFFS (DIFFS CUR GOL))
   LOOP (SETQ XDIFF (LIST (CAR NDIFFS)))
        (COND ((NULL XDIFF) (RETURN '(NIL (DONE))))
              ((SETQ CANPEG (ANY1S XDIFF (ALL_1MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              ((SETQ CANPEG (ANY2S XDIFF (ALL_2MOVES CUR)))
               (RETURN (MAKE CANPEG CUR)))
              (T (SETQ CANORDER (REVERSE CANORDER))
                 (SETQ MV (MOD9 CUR GOL))
                 (SETQ CANORDER (REVERSE CANORDER))
                 (RETURN MV)))))

(DEFUN MOVE (CAN TO STATE)
  (COMMENT MOVE UNCONDITIONALLY AND RETURN NEW STATE)
  ((LAMBDA (POF)
     (LIST (CONS POF (PICKUP CAN STATE))
           (CONS TO (PLACE CAN TO STATE))
           ((LAMBDA (OTH) (CONS OTH (SEEPEG OTH STATE)))
            (OTHER POF TO))))
   (PEGOF CAN STATE)))

(DEFUN NEXTMOVE (FN CURRENT GOAL)
  (COMMENT RETURN NEXT MOVE FROM CURRENT TO GOAL USING FN)
  (PROG (CAN FROM TO CANTO)
    (SETQ CANTO (CAR (FUNCALL FN CURRENT GOAL)))
    (SETQ CAN (CAR CANTO))
    (SETQ TO (CADR CANTO))
    (SETQ FROM (PEGOF CAN CURRENT))
    (RETURN (LIST CAN (LIST (STRIPP FROM) (STRIPP TO))))))

(DEFUN NEXTSTATES (STATE)
  (COMMENT RETURN LIST OF ALL STATES 1. MOVE FROM CURRENT STATE)
  (MAPCAR '(LAMBDA (MV) (MOVE (CAR MV) (CADR MV) STATE))
          (LEGAL_MOVES STATE)))

;;;NTHPROB useful for selecting a particular problem from PROBSTATES.
;;;     do (SETUP) before using.
;;;

(DEFUN NTHPROB (N) (CADAR (NTHCDR (DIFFERENCE N 9.) PROBSTATES)))

(DEFUN OKORDER (CANX CANY) (BEFORE CANX CANY '(C3 C2 C1)))

(DEFUN OTHER (X Y) (CAR (REMAINS (LIST X Y) '(PEGA PEGB PEGC))))

(DEFUN PEGOF (CAN STATE)
  (COMMENT RETURN PEG HOLDING CAN IN STATE)
  (COND ((NULL STATE) (PRINT '|err: cant find can|))
        ((MEMBER CAN (CDAR STATE)) (CAAR STATE))
        (T (PEGOF CAN (CDR STATE)))))

(DEFUN PICKUP (CAN STATE)
  (COMMENT RETURN LIST OF PEG SANS CAN)
  (REMOVE CAN (SEEPEG (PEGOF CAN STATE) STATE)))
```

```
(DEFUN PLACE (CAN PEG STATE)
  (COMMENT RETURN LIST OF CAN ADDED TO TOP OF PEGS CAN LIST)
  (CONS CAN (SEEPEG PEG STATE)))

(DEFUN REMAINS (SET1 SET2)
  (COMMENT REMOVES ALL MEMBS OF SET1 FM SET2)
  (COND ((NULL SET1) SET2)
        (T (REMAINS (CDR SET1) (REMOVE (CAR SET1) SET2)))))

(DEFUN REMOVE (X L)
  (COMMENT REMOVE ALL OCCURRENCES OF X FM L)
  (COND ((NULL L) NIL)
        ((EQ (CAR L) X) (REMOVE X (CDR L)))
        (T (CONS (CAR L) (REMOVE X (CDR L))))))

(DEFUN SEEPEG (PEG STATE)
  (COMMENT RETURN LIST OF CANS ON PEG IN STATE)
  (CDR (ASSOC PEG STATE)))

(DEFUN SETUP NIL
  (SETQ TOWA '((PEGA C3 C2 C1) (PEGB) (PEGC)))
  (SETQ TOWB '((PEGA) (PEGB C3 C2 C1) (PEGC)))
  (SETQ FLAT1 '((PEGA C1) (PEGB C2) (PEGC C3)))
  (SETQ FLAT2 '((PEGA C2) (PEGB C1) (PEGC C3)))
  (SETUP2))

(DEFUN SETUP2 NIL
  (TRACEOFF)
  (SETQ CANORDER '(C3 C2 C1))
  (SETQ MOVELIST
        '((C1 PEGA)
          (C1 PEGB)
          (C1 PEGC)
          (C2 PEGA)
          (C2 PEGB)
          (C2 PEGC)
          (C3 PEGA)
          (C3 PEGB)
          (C3 PEGC)))
  (SLURP TOHDAT)
  (SETPROB)
  (PRINT 'PROBLEMS_LOADED))

(DEFUN SOLVE (FN INIT FIN)
  (COMMENT SOLVE FROM INIT TO FIN USING FN)
  (COMMENT PRINT MOVES AND STATES ,SAVE LIST OF BOTH)
  (PROG (NEWSTATE MOVESTATE MOVENO STATELIST)
        (SETQ MOVENO 0.)
        (SETQ MOVESTATE (SETQ NEWSTATE INIT))
   LOOP (SETQ STATELIST (CONS MOVESTATE STATELIST))
        (SETQ MOVESTATE (FUNCALL FN NEWSTATE FIN))
        (COND ((EQ (CAADR MOVESTATE) 'DONE)
               (RETURN (COND (TRACEON (REVERSE STATELIST))
                             (T 'SOLVED))))
              (T (PRINT (SETQ MOVENO (1+ MOVENO)))
                 (PRIN1 (CAR MOVESTATE))
                 (PRIN1 (SETQ NEWSTATE (CADR MOVESTATE)))
                 (GO LOOP)))))

(DEFUN SOLVED (STATE GOAL)
  (COMMENT ASSUME STATE IS LEGAL ,TEST FOR SOLUTION)
  (COND ((NULL (DIFFS STATE GOAL)))
        (T NIL)))

(DEFUN STRIPP (PEGNAME) (NTH 3. (EXPLODE PEGNAME)))

(DEFUN TOPCAN (PEG STATE) (CAR (SEEPEG PEG STATE)))

(DEFUN TRACEOFF NIL (SETQ TRACEON NIL))

(DEFUN TRACEON NIL (SETQ TRACEON T))

(SETQ TRACEON NIL)

(DEFUN TRY1O (CAN TO STATE)
  (COMMENT TRY SPECIFIED MOVE ,USING SPH PERC STRAT ,RETURN MOVE
           ACTUALLY SELECTED AND NEW STATE)
  (COND (TRACEON
         (PRINT '|trying to move|)
         (PRIN1 CAN)
```

A-8

```
        (PRIN1 '| to |)
        (PRIN1 TO)))
  (COND ((LEGAL CAN TO STATE) (MAKE (LIST CAN TO) STATE))
        (T (TRY10 (CULP10 CAN TO STATE)
                  (OTHER (PEGOF CAN STATE) TO)
                  STATE))))

(DEFUN TRY2 (CAN TO STATE)
  (COMMENT NOTE NEW STATE NOT RETURNED .ONLY MOVE)
  (PROG (TOP FROM)
    (SETQ FROM (PEGOF CAN STATE))
    (SETQ TOP (TOPCAN FROM STATE))
    (COND ((EQ TOP CAN) (RETURN (LIST (LIST CAN TO))))
          (T (RETURN (LIST (LIST TOP (EMPTY STATE))))))))

;;;TRY3 and TRY5 used by MOD3 and MOD4
;;;

(DEFUN TRY3 (CAN TO STATE)
  (COMMENT TRY SPECIFIED MOVE .IF BLOCKED RETURN MOVE THAT CLEARS TO
           PEG)
  (COND ((LEGAL CAN TO STATE) (LIST CAN TO))
        (T (COND ((NOT (OKORDER CAN (TOPCAN TO STATE)))
                  (LIST (TOPCAN TO STATE)
                        (OTHER (PEGOF CAN STATE) TO)))
                 (T (LIST (TOPCAN (PEGOF CAN STATE) STATE)
                          (OTHER (PEGOF CAN STATE) TO)))))))

(DEFUN TRY4 (CAN TO STATE)
  (COMMENT TRY SPECIFIED MOVE .IF BLOCKED RETURN MOVE THAT CLEARS FROM
           PEG)
  (COND ((LEGAL CAN TO STATE) (LIST CAN TO))
        (T (COND ((NOT (EQ CAN (TOPCAN (PEGOF CAN STATE) STATE)))
                  (LIST (TOPCAN (PEGOF CAN STATE) STATE)
                        (OTHER (PEGOF CAN STATE) TO)))
                 (T (LIST (TOPCAN TO STATE)
                          (OTHER (PEGOF CAN STATE) TO)))))))

(DEFUN MODPROB (FN N)
  (COMMENT APPLY MODEL FN TO PROBLEM N)
  (PRINT (SETQ N (NTHPROB N)))
  (TERPRI)
  (FUNCALL FN (CAR N) (CADR N)))
```

A-9

```
|Dribbling.|

;;;Some examples of how to use the models from Klahr & Robinson
;;;
;;;
;;;  To get started:

(slurp toh5)(A310DK17 TOH5 MCL)

(setup)
PROBLEMS_LOADED T

;;;Now the criterial problemset has been loaded, and a few tower and
;;; flat states have been created
;;; The full problems set is a list named PROBSTATES
towa
((PEGA C3 C2 C1) (PEGB) (PEGC))

flat2
((PEGA C2) (PEGB C1) (PEGC C3))


;;;to specify a problem, take car and caadr of nthprob, eg:
(nthprob 13)(((PEGA) (PEGB C3 C2) (PEGC C1)) ((PEGA C3 C2 C1) (PEGB) (PEGC)))

(car (nthprob 13))((PEGA) (PEGB C3 C2) (PEGC C1))
(cadr (nthprob 13))((PEGA C3 C2 C1) (PEGB) (PEGC))

;
;to make the next move from a particular state, using a particular function
;use NEXTMOVE with function name (quoted) and initial and final states
(nextmove 'mod3 flat1 towa)(C2 (B A))

(nextmove 'mod10 flat1 towa)(C2 (B A))

;
;
;to repeatedly apply model to sequential states produced by that model,
;use SOLVE
(setq p20 (nthprob 20))(((PEGA C1) (PEGB C2) (PEGC C3)) ((PEGA) (PEGB) (PEGC C3 C2 C1)))

(solve 'mod10 (car p20)(cadr p20))
1 (C3 PEGB)((PEGC) (PEGB C3 C2) (PEGA C1))
2 (C1 PEGC)((PEGA) (PEGC C1) (PEGB C3 C2))
3 (C3 PEGA)((PEGB C2) (PEGA C3) (PEGC C1))
4 (C2 PEGC)((PEGB) (PEGC C2 C1) (PEGA C3))
5 (C8 PEGC)((PEGA) (PEGC C3 C2 C1) (PEGB))SOLVED

;
;to apply model to problem for single move, use MODPROB
(modprob 'mod6 20)
(((PEGA C1) (PEGB C2) (PEGC C3)) ((PEGA) (PEGB) (PEGC C3 C2 C1)))
((C3 PEGB) ((PEGC) (PEGB C3 C2) (PEGA C1)))

;;output is initial-final state pair, follwed by move and new state
;
;
;to apply a model (or models) to all the criterial problems, edit the
;two lists in ALLMOVES and execute it
(editf allmoves)
EDIT
#
f list
(LIST 1 2 3 4 5 6 7 8 9)
#
0
(MAPC '(LAMBDA # ...) (LIST 1 ...))
#
f list
(LIST 1 2 3 4 5 6 7 8 9)
#
(3)(3)(3)(3)(3)
(LIST 1 7 8 9)
#
f
(LIST 'MOD1 'MOD2 'MOD3 'MOD4 'MOD5 'MOD6 'MOD7 'MOD8B 'MOD10)
#
(3)(3)(3)(3)(3)
(LIST 'MOD1 'MOD7 'MOD8B 'MOD10)
#
ekALLMOVES
```

A-10

```
(allmoves)
PB     MOD1    MOD7    MOD8    MOD9
9    C2_CB   C3_CA   C3_CA   C3_CA
10    C2_CA   C3_CB   C3_CB   C3_CB
11    C2_AB   C3_AC   C3_AC   C3_AC
12    C2_BC   C3_BA   C3_BA   C3_BA
13    C1_CA   C1_CA   C1_CA   C1_CA
14    C1_BA   C1_BA   C1_BA   C1_BA
15    C1_BA   C1_BA   C1_BA   C1_BA
16    C1_CA   C3_AB   C2_CB   C2_CB
17    C1_CA   C3_CB   C3_CB   C3_CB
18    C1_CA   C3_CB   C3_CA   C3_CA
19    C1_AC   C3_AB   C3_AC   C3_AC
20    C1_AC   C3_CB   C3_CB   C3_CB
21    C1_CA   C3_AB   C3_AC   C3_AC
22    C1_AC   C3_AB   C3_AB   C3_AB
23    C1_CA   C3_AB   C3_AB   C3_AB
24    C1_AC   C3_CB   C3_CA   C3_CA
25    C1_BA   C3_AC   C2_BC   C2_BC
26    C1_BA   C3_AC   C2_BC   C2_BC
27    C1_BC   C3_CA   C3_CA   C3_CB
28    C1_AC   C3_CB   C2_AB   C2_AB
29    C1_CA   C3_AB   C2_CB   C2_CB
30    C1_BC   C3_BA   C2_CA   C2_CA
31    C1_BC   C3_CA   C2_BA   C2_BA
32    C1_BA   C3_AC   C3_AC   C3_AB
33    C1_BA   C3_BC   C3_BC   C3_BA
34    C1_BC   C3_BA   C3_BA   C3_BC
35    C1_BA   C3_CA   C2_AB   C3_CB
36    C1_BC   C3_AC   C2_CB   C3_AB
37    C1_AC   C3_BC   C3_BC   C3_BA
38    C1_BA   C3_BC   C3_BC   C3_BA
39    C1_BA   C3_CA   C3_CA   C3_CB
40    C1_BC   C3_BA   C3_BA   C3_BC
DONE

;;;
;;;
;;;for garralous output, turn trace on
(traceon)T

(solve 'mod10 towa towb)
|trying to move| C1| to |PEGB
|can not free to move|
|trying to move| C2| to |PEGC
|can not free to move|
|trying to move| C3| to |PEGB
1 (C3 PEGB)((PEGA C2 C1) (PEGB C3) (PEGC))
|trying to move| C1| to |PEGB
|can not free to move|
|trying to move| C2| to |PEGC
2 (C2 PEGC)((PEGA C1) (PEGC C2) (PEGB C3))
|trying to move| C1| to |PEGB
|destination peg blocked|
|trying to move| C3| to |PEGC
3 (C3 PEGC)((PEGB) (PEGC C3 C2) (PEGA C1))
|trying to move| C1| to |PEGB
4 (C1 PEGB)((PEGA) (PEGB C1) (PEGC C3 C2))
|trying to move| C2| to |PEGB
|can not free to move|
|trying to move| C3| to |PEGA
5 (C3 PEGA)((PEGC C2) (PEGA C3) (PEGB C1))
|trying to move| C2| to |PEGB
6 (C2 PEGB)((PEGC) (PEGB C2 C1) (PEGA C3))
|trying to move| C3| to |PEGB
7 (C3 PEGB)((PEGA) (PEGB C3 C2 C1) (PEGC))(((PEGA C3 C2 C1)
                                             (PEGB)
                                             (PEGC))
                                            ((C3 PEGB)
                                             ((PEGA C2 C1)
                                              (PEGB C3)
                                              (PEGC)))
                                            ((C2 PEGC)
                                             ((PEGA C1)
                                              (PEGC C2)
                                              (PEGB C3)))
                                            ((C3 PEGC)
                                             ((PEGB)
                                              (PEGC C3 C2)
                                              (PEGA C1)))
```

B-1

```
(COMMENT --CONTENTS-- WMPSIZE PROB1 PROB2 )

(SLURP C380ML5P UCEDIT)
(slurp jill)


(SETQ WMPSIZE '10.)

(SWITCHES KEEP-LHS ON TRACE LEVEL2)

(system
q1tq3
 ( (q1) & =e1 --> (<delete> =e1)(q3))


q3toq4
 ( (q3) & =e1 --> (<delete> =e1)(q4))

solved
( (q3) - (want ! =) --> (<write>)(<write> problem solved)(<write>)(<halt>))


topgoal
((q1)(TOPGOAL =PA =PB =PC) & =e1 --> (<delete> =e1)
    (attended =e1)(GOAL =PA) (GOAL =PB) (GOAL =PC) (a isa peg)
                (b isa peg)(c isa peg))

EMPTYGOAL
((q1)(GOAL (=)) & =E1 --> (<DELETE> =E1))

goalcan
((q1) (goal (peg =peg =cx ! =y)) & =e1 -->
                (<delete> =e1)(goal =cx on =peg)(goal (peg =peg ! =y)))
unpack
((q1)(STATE =PA =PB =PC) --> =PA =PB =PC (3 BIGGER 2) (3 BIGGER 1) (3 BIGGER 0)
(2 BIGGER 1) (2 BIGGER 1) (2 BIGGER 0) (1 BIGGER 0))

seenone
((q1)(peg =peg) --> (see topof =peg IS 0))

see1
((q1)(peg =peg =CX) --> (see =CX ON =peg) (see topof =peg IS =CX))

see2
((q1)(peg =peg =CX =CY) --> (see =CX on =peg) (see =CY on =peg) (see topof  =peg IS
=CX) (see =cx above =CY))

see3
((q1)(peg =peg =CX =CY =CZ) --> (see =CX on =peg) (see =CY on =peg) (see =CZ ON
=PEG) (see topof =PEG IS =CX) (see =cx above =CY) (see =cx above =CZ) (see  =cy above  =CZ))


DIFF
((q3)(GOAL =CAN ON =PEG1) (see =CAN ON #PEG1 & =PEG2)  -->
(WANT (=CAN =PEG2 =PEG1)))


WANT2
((q4)(want (=CAN1 ! =)) & =E1 (WANT (=CAN2 ! =)) & =E2 (=CAN1 BIGGER =CAN2) -->
(<DELETE> =E1))
TRY
((q4) & =e0 (want (=CAN =PEG1 =PEG2)) & =E1  --> (<DELETE> =e0  =E1) (q5)
(TRY =CAN =PEG1 =PEG2))

LEGAL
((q5) & =e0 (try =CAN1 =P1 =P2) & =E1 (see topof =P1 IS =CAN1) (see topof =P2 IS =CAN2)
(=CAN1 BIGGER =CAN2) -->(q6) (MOVE =CAN1 =P1 =P2) (<DELETE> =e0 =e1))

TOBLOCK
((q5) (try =CAN1 =P1 =P2) & =e1   (see topof =P2 IS =CAN2) (=CAN2 BIGGER =CAN1)
--> (=CAN2 BLOCKS =E1))

FROMBLOCK
((q5) (try =CAN1 =P1 =P2) & =e1  (see =CAN2 ABOVE =CAN1) -->
 (=CAN2 BLOCKS =E1))

TRIED
((q5) & =E1 (try ! =) & =e2 --> (<DELETE> =E1 =e2 ) (q5a))
```



```
MINBLOCKER
```

B-2

```
((q5)   (=CAN1 BLOCKS =MV) & =e2 (=CAN2 BLOCKS =MV) (=CAN1 BIGGER =CAN2) -->
(<DELETE>  =e2))

subtry
((q5a) & =e1 (=C1 BLOCKS (try =C2 =P1 =P2) & =E3) & =E2
        (=p4 & #p1 & #p2 isa peg)
 (see =C1 ON =p3) --> (<DELETE> =e3 =E2 =E1)  (try
=C1 =p3 =P4) (q5))

prep
((q6)(MOVE ! =) (see ! =) & =E1 --> (<DELETE> =E1))

move
((q6) & =e0 (MOVE =CAN =P1 =P2) & =E1 (PEG =P2 ! =TOCANS) & =E2 (PEG =P1 =CAN !
=FROMCANS) & =E3 (PEG ! =P3) & #E3 & =E4 & #E2 (STATE ! =) & =E5 -->
(<DELETE> =e0  =E1 =E2 =E3 =E4 =E5) (STATE (PEG =P1 ! =FROMCANS) (PEG =P2 =CAN !
=TOCANS) (PEG ! =P3))(q1))



))))))))))))))))

(DEFUN PROBSET (INITIAL FINAL)
  (LIST '(q1) (CONS 'STATE INITIAL) (CONS 'TOPGOAL FINAL)))
(setq towa '( (peg a 3 2 1)(peg b)(peg c)))
(setq towb '((peg a)(peg b 3 2 1)(peg c)))
(setq flat1 '((peg a 3)(peg b 2)(peg c 1)))
(setq flat2 '((peg a 3)(peg b 1)(peg c 2)))

(SETQ PROB1
    '((q1)(state (PEG A) (PEG B 3. 1.) (PEG C 2.))
      (TOPGOAL (peg A 3. 2. 1.) (peg B) (peg C))))

(SETQ PROB2
      '((q1)(state (PEG A) (PEG B 3. 1.) (PEG C 2.))
        (TOPGOAL (peg a 2.) (peg b 1.) (peg c 3.))))

(setq prob3
     '((q1)(state (peg a) (peg b 3 2 1)(peg c ))
    (topgoal (peg a 3 2 1)(peg b)(peg c))))
```

B-3

```
|Dribbling.|
β
ppβ
((Q1)
 (STATE (PEG A 3) (PEG B 2) (PEG C 1))
 (TOPGOAL (PEG A 3) (PEG B 1) (PEG C 2)))

β
::::seven move flat to flatβ
β
(start pp)Warning: WM and the network memory may be inconsistent

>>
  run 60 terseβ 1. UNPACK
        2. SEE1 3. SEE1 4. SEE1 5. TOPGOAL      6. GOALCAN      7. GOALCAN
        8. GOALCAN      9. Q1TQ3
10. DIFF        11. DIFF        12. Q3TOQ4      13. WANT2       14. TRY 15. TO
BLOCK   16. TRIED       17. SUBTRY      18. TOBLOCK     19. TRIED
20. SUBTRY      21. LEGAL       22. PREP        23. PREP        24. PREP
        25. PREP        26. PREP        27. PREP        28. MOVE        29. UNPACK
30. SEENONE     31. SEE2        32. SEE1        33. Q1TQ3       34. DIFF
        35. DIFF        36. DIFF        37. Q3TOQ4      38. WANT2       39. WANT2
40. TRY 41. TOBLOCK     42. FROMBLOCK   43. MINBLOCKER  44. TRIED       45. SU
BTRY    46. LEGAL       47. PREP        48. PREP        49. PREP
50. PREP        51. PREP        52. PREP        53. PREP        54. MOVE
        55. UNPACK      56. SEENONE     57. SEE1        58. SEE2        59. Q1TQ3
60. DIFF
>>
  (wm)
((STATE (PEG B) (PEG A 2) (PEG C 3 1)) (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1)
(3 BIGGER 0) (3 BIGGER 1) (3 BIGGER 2) (PEG C 3 1) (PEG A 2) (PEG B) (SEE
TOPOF B IS 0) (SEE TOPOF A IS 2) (SEE 2 ON A) (SEE 3 ABOVE 1) (SEE TOPOF C
IS 3) (SEE 1 ON C) (SEE 3 ON C) (C ISA PEG) (B ISA PEG) (A ISA PEG) (GOAL
(PEG A)) (GOAL 3 ON A) (GOAL (PEG B)) (GOAL 1 ON B) (GOAL (PEG C)) (GOAL 2
ON C) (ATTENDED (TOPGOAL (PEG A 3) (PEG B 1) (PEG C 2))) (Q3) (WANT (3 C
A)))
>>
run 100 fullβ

DIFF
  (SEE 1 ON C) (GOAL 1 ON B) (Q3)
  -->
   (WANT (1 C B)) .


DIFF
  (SEE 2 ON A) (GOAL 2 ON C) (Q3)
  -->
   (WANT (2 A C))

Q3TOQ4
  (Q3)
  -->
   (Q4)
  Deleted:
   (Q3)

WANT2
  (2 BIGGER 1) (WANT (1 C B)) (WANT (2 A C)) (Q4)
  -->
  Deleted:
   (WANT (2 A C))

WANT2
  (3 BIGGER 1) (WANT (1 C B)) (WANT (3 C A)) (Q4)
  -->
  Deleted:
   (WANT (3 C A))

TRY
  (WANT (1 C B)) (Q4)
  -->
   (TRY 1 C B) (Q5)
  Deleted:
   (WANT (1 C B)) (Q4)
```

B-4

```
FROMBLOCK
  (SEE 3 ABOVE 1) (TRY 1 C B) (Q5)
  -->
   (3 BLOCKS (TRY 1 C B))


TRIED
  (TRY 1 C B) (Q5)
  -->
   (Q5A)
  Deleted:
   (TRY 1 C B) (Q5)


SUBTRY
  (SEE 3 ON C) (A ISA PEG) (3 BLOCKS (TRY 1 C B)) (Q5A)
  -->
   (Q5) (TRY 3 C A)
  Deleted:
   (Q5A) (3 BLOCKS (TRY 1 C B))


LEGAL
  (3 BIGGER 2) (SEE TOPOF A IS 2) (SEE TOPOF C IS 3) (TRY 3 C A) (Q5)
  -->
   (MOVE 3 C A) (Q6)
  Deleted:
   (TRY 3 C A) (Q5)


PREP
  (SEE 3 ON C) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE 3 ON C)

PREP
  (SEE 1 ON C) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE 1 ON C)

PREP
  (SEE TOPOF C IS 3) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE TOPOF C IS 3)


PREP
  (SEE 3 ABOVE 1) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE 3 ABOVE 1)


PREP
  (SEE 2 ON A) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE 2 ON A)


PREP
  (SEE TOPOF A IS 2) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE TOPOF A IS 2)


PREP
  (SEE TOPOF B IS 0) (MOVE 3 C A) (Q6)
  -->
  Deleted:
   (SEE TOPOF B IS 0)
```



```
MOVE
```

B-5

```
  (STATE (PEG B) (PEG A 2) (PEG C 3 1)) (PEG B) (PEG C 3 1) (PEG A 2)
  (MOVE 3 C A) (Q6)
  -->
   (Q1) (STATE (PEG C 1) (PEG A 3 2) (PEG B))
  Deleted:
   (STATE (PEG B) (PEG A 2) (PEG C 3 1)) (PEG B) (PEG C 3 1) (PEG A 2)
   (MOVE 3 C A) (Q6)


UNPACK
  (STATE (PEG C 1) (PEG A 3 2) (PEG B)) (Q1)
  -->
   (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1) (3 BIGGER 0) (3 BIGGER 1)
   (3 BIGGER 2) (PEG B) (PEG A 3 2) (PEG C 1)


SEE1
  (PEG C 1) (Q1)
  -->
   (SEE TOPOF C IS 1) (SEE 1 ON C)


SEE2
  (PEG A 3 2) (Q1)
  -->
   (SEE 3 ABOVE 2) (SEE TOPOF A IS 3) (SEE 2 ON A) (SEE 3 ON A)


SEENONE
  (PEG B) (Q1)
  -->
   (SEE TOPOF B IS 0)


Q1TQ3
  (Q1)
  -->
   (Q3)
  Deleted:
   (Q1)


DIFF
  (SEE 2 ON A) (GOAL 2 ON C) (Q3)
  -->
   (WANT (2 A C))


DIFF
  (SEE 1 ON C) (GOAL 1 ON B) (Q3)
  -->
   (WANT (1 C B))


Q3TOQ4
  (Q3)
  -->
   (Q4)
  Deleted:
   (Q3)


WANT2
  (2 BIGGER 1) (WANT (1 C B)) (WANT (2 A C)) (Q4)
  -->
  Deleted:
   (WANT (2 A C))


TRY
  (WANT (1 C B)) (Q4)
  -->
   (TRY 1 C B) (Q5)
  Deleted:
   (WANT (1 C B)) (Q4)


LEGAL
  (1 BIGGER 0) (SEE TOPOF B IS 0) (SEE TOPOF C IS 1) (TRY 1 C B) (Q5)
  -->
   (MOVE 1 C B) (Q6)
```

B-C

```
  Deleted:
   (TRY 1 C B) (Q5)

PREP
  (SEE TOPOF B IS 0) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE TOPOF B IS 0)

PREP
  (SEE 3 ON A) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE 3 ON A)

PREP
  (SEE 2 ON A) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE 2 ON A)

PREP
  (SEE TOPOF A IS 3) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE TOPOF A IS 3)

PREP
  (SEE 3 ABOVE 2) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE 3 ABOVE 2)

PREP
  (SEE 1 ON C) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE 1 ON C)

PREP
  (SEE TOPOF C IS 1) (MOVE 1 C B) (Q6)
  -->
  Deleted:
   (SEE TOPOF C IS 1)

MOVE
  (STATE (PEG C 1) (PEG A 3 2) (PEG B)) (PEG A 3 2) (PEG C 1) (PEG B)
  (MOVE 1 C B) (Q6)
  -->
   (Q1) (STATE (PEG C) (PEG B 1) (PEG A 3 2))
  Deleted:
   (STATE (PEG C 1) (PEG A 3 2) (PEG B)) (PEG A 3 2) (PEG C 1) (PEG B)
   (MOVE 1 C B) (Q6)

UNPACK
  (STATE (PEG C) (PEG B 1) (PEG A 3 2)) (Q1)
  -->
   (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1) (3 BIGGER 0) (3 BIGGER 1)
   (3 BIGGER 2) (PEG A 3 2) (PEG B 1) (PEG C)

SEENONE
  (PEG C) (Q1)
  -->
   (SEE TOPOF C IS 0)

SEE1
  (PEG B 1) (Q1)
  -->
   (SEE TOPOF B IS 1) (SEE 1 ON B)
```

```
SEE2
  (PEG A 3 2) (Q1)
  -->
   (SEE 3 ABOVE 2) (SEE TOPOF A IS 3) (SEE 2 ON A) (SEE 3 ON A)


Q1TQ3
  (Q1)
  -->
   (Q3)
  Deleted:
   (Q1)


DIFF
  (SEE 2 ON A) (GOAL 2 ON C) (Q3)
  -->
   (WANT (2 A C))


Q3TOQ4
  (Q3)
  -->
   (Q4)
  Deleted:
   (Q3)


TRY
  (WANT (2 A C)) (Q4)
  -->
   (TRY 2 A C) (Q5)
  Deleted:
   (WANT (2 A C)) (Q4)


FROMBLOCK
  (SEE 3 ABOVE 2) (TRY 2 A C) (Q5)
  -->
   (3 BLOCKS (TRY 2 A C))


TRIED
  (TRY 2 A C) (Q5)
  -->
   (Q5A)
  Deleted:
   (TRY 2 A C) (Q5)


SUBTRY
  (SEE 3 ON A) (B ISA PEG) (3 BLOCKS (TRY 2 A C)) (Q5A)
  -->
   (Q5) (TRY 3 A B)
  Deleted:
   (Q5A) (3 BLOCKS (TRY 2 A C))


LEGAL
  (3 BIGGER 1) (SEE TOPOF B IS 1) (SEE TOPOF A IS 3) (TRY 3 A B) (Q5)
  -->
   (MOVE 3 A B) (Q6)
  Deleted:
   (TRY 3 A B) (Q5)


PREP
  (SEE 3 ON A) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE 3 ON A)


PREP
  (SEE 2 ON A) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE 2 ON A)
```

```
PREP
  (SEE TOPOF A IS 3) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE TOPOF A IS 3)

PREP
  (SEE 3 ABOVE 2) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE 3 ABOVE 2)

PREP
  (SEE 1 ON B) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE 1 ON B)

PREP
  (SEE TOPOF B IS 1) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE TOPOF B IS 1)

PREP
  (SEE TOPOF C IS 0) (MOVE 3 A B) (Q6)
  -->
  Deleted:
   (SEE TOPOF C IS 0)

MOVE
  (STATE (PEG C) (PEG B 1) (PEG A 3 2)) (PEG C) (PEG A 3 2) (PEG B 1)
  (MOVE 3 A B) (Q6)
  -->
   (Q1) (STATE (PEG A 2) (PEG B 3 1) (PEG C))
  Deleted:
   (STATE (PEG C) (PEG B 1) (PEG A 3 2)) (PEG C) (PEG A 3 2) (PEG B 1)
   (MOVE 3 A B) (Q6)

UNPACK
  (STATE (PEG A 2) (PEG B 3 1) (PEG C)) (Q1)
  -->
   (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1) (3 BIGGER 0) (3 BIGGER 1)
   (3 BIGGER 2) (PEG C) (PEG B 3 1) (PEG A 2)

SEE1
  (PEG A 2) (Q1)
  -->
   (SEE TOPOF A IS 2) (SEE 2 ON A)

SEE2
  (PEG B 3 1) (Q1)
  -->
   (SEE 3 ABOVE 1) (SEE TOPOF B IS 3) (SEE 1 ON B) (SEE 3 ON B)

SELNONE
  (PEG C) (Q1)
  -->
   (SEE TOPOF C IS 0)

Q1TQ3
  (Q1)
  -->
   (Q3)
  Deleted:
   (Q1)

DIFF
  (SEE 3 ON B) (GOAL 3 ON A) (Q3)
  -->
```

B-9

```
  (WANT (3 B A))

DIFF
  (SEE 2 ON A) (GOAL 2 ON C) (Q3)
  -->
  (WANT (2 A C))

Q3TOQ4
  (Q3)
  -->
  (Q4)
  Deleted:
  (Q3)

WANT2
  (3 BIGGER 2) (WANT (2 A C)) (WANT (3 B A)) (Q4)
  -->
  Deleted:
  (WANT (3 B A))

TRY
  (WANT (2 A C)) (Q4)
  -->
  (TRY 2 A C) (Q5)
  Deleted:
  (WANT (2 A C)) (Q4)

LEGAL
  (2 BIGGER 0) (SEE TOPOF C IS 0) (SEE TOPOF A IS 2) (TRY 2 A C) (Q5)
  -->
  (MOVE 2 A C) (Q6)
  Deleted:
  (TRY 2 A C) (Q5)

PREP
  (SEE TOPOF C IS 0) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE TOPOF C IS 0)

PREP
  (SEE 3 ON B) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE 3 ON B)

PREP
  (SEE 1 ON B) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE 1 ON B)

PREP
  (SEE TOPOF B IS 3) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE TOPOF B IS 3)

PREP
  (SEE 3 ABOVE 1) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE 3 ABOVE 1)

PREP
  (SEE 2 ON A) (MOVE 2 A C) (Q6)
  -->
  Deleted:
  (SEE 2 ON A)
```

B-10

```
PREP
  (SEE TOPOF A IS 2) (MOVE 2 A C) (Q6)
  -->
  Deleted:
   (SEE TOPOF A IS 2)

MOVE
  (STATE (PEG A 2) (PEG B 3 1) (PEG C)) (PEG B 3 1) (PEG A 2) (PEG C)
  (MOVE 2 A C) (Q6)
  -->
   (Q1) (STATE (PEG A) (PEG C 2) (PEG B 3 1))
  Deleted:
   (STATE (PEG A 2) (PEG B 3 1) (PEG C)) (PEG B 3 1) (PEG A 2) (PEG C)
   (MOVE 2 A C) (Q6)

UNPACK
  (STATE (PEG A) (PEG C 2) (PEG B 3 1)) (Q1)
  -->
   (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1) (3 BIGGER 0) (3 BIGGER 1)
   (3 BIGGER 2) (PEG B 3 1) (PEG C 2) (PEG A)

SEENONE
  (PEG A) (Q1)
  -->
   (SEE TOPOF A IS 0)

SEE1
  (PEG C 2) (Q1)
  -->
   (SEE TOPOF C IS 2) (SEE 2 ON C)

SEE2
  (PEG B 3 1) (Q1)
  -->
   (SEE 3 ABOVE 1) (SEE TOPOF B IS 3) (SEE 1 ON B) (SEE 3 ON B)

Q1TQ3
  (Q1)
  -->
   (Q3)
  Deleted:
   (Q1)

DIFF
  (SEE 3 ON B) (GOAL 3 ON A) (Q3)
  -->
   (WANT (3 B A))

Q3TOQ4
  (Q3)
  -->
   (Q4)
  Deleted:
   (Q3)

TRY
  (WANT (3 B A)) (Q4)
  -->
   (TRY 3 B A) (Q5)
  Deleted:
   (WANT (3 B A)) (Q4)

LEGAL
  (3 BIGGER 0) (SEE TOPOF A IS 0) SEE TOPOF B IS 3) (TRY 3 B A) (Q5)
  -->
   (MOVE 3 B A) (Q6)
  Deleted:
   (TRY 3 B A) (Q5)
```

B-11

```
PREP
  (SEE 3 ON B) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE 3 ON B)

PREP
  (SEE 1 ON B) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE 1 ON B)

PREP
  (SEE TOPOF B IS 3) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE TOPOF B IS 3)

PREP
  (SEE 3 ABOVE 1) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE 3 ABOVE 1)

PREP
  (SEE 2 ON C) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE 2 ON C)

PREP
  (SEE TOPOF C IS 2) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE TOPOF C IS 2)

PREP
  (SEE TOPOF A IS 0) (MOVE 3 B A) (Q6)
  -->
  Deleted:
   (SEE TOPOF A IS 0)

MOVE
  (STATE (PEG A) (PEG C 2) (PEG B 3 1)) (PEG C 2) (PEG B 3 1) (PEG A)
  (MOVE 3 B A) (Q6)
  -->
  (Q1) (STATE (PEG B 1) (PEG A 3) (PEG C 2))
  Deleted:
   (STATE (PEG A) (PEG C 2) (PEG B 3 1)) (PEG C 2) (PEG B 3 1) (PEG A)
   (MOVE 3 B A) (Q6)

UNPACK
  (STATE (PEG B 1) (PEG A 3) (PEG C 2)) (Q1)
  -->
   (1 BIGGER 0) (2 BIGGER 0) (2 BIGGER 1) (3 BIGGER 0) (3 BIGGER 1)
   (3 BIGGER 2) (PEG C 2) (PEG A 3) (PEG B 1)

SEE1
  (PEG B 1) (Q1)
  -->
   (SEE TOPOF B IS 1) (SEE 1 ON B)

SEE1
  (PEG A 3) (Q1)
  -->
   (SEE TOPOF A IS 3) (SEE 3 ON A)

SEE1
  (PEG C 2) (Q1)
  -->
```

B-12

```
   (SEE TOPOF C IS 2) (SEE 2 ON C)

Q1TQ3
  (Q1)
  -->
  (Q3)
  Deleted:
  (Q1)

 PROBLEM SOLVED

SOLVED
  (Q3)
  -->

END -- EXPLICIT HALT
 22 productions (160 / 275 nodes) (75 / 163 features)
 159 firings (427 RHS actions)
 25.855346 mean working memory size (31 maximum)
 3.02515724 mean conflict set size (8 maximum)
 39.735849 mean token memory size (61 maximum)
 6.966 seconds (43.811321 msec per firing) (16.3138173 msec per action)
NIL

(wm)((STATE (PEG B 1) (PEG A 3) (PEG C 2))
 (1 BIGGER 0)
 (2 BIGGER 0)
 (2 BIGGER 1)
 (3 BIGGER 0)
 (3 BIGGER 1)
 (3 BIGGER 2)
 (PEG C 2)
 (PEG A 3)
 (PEG B 1)
 (SEE TOPOF B IS 1)
 (SEE 1 ON B)
 (SEE TOPOF A IS 3)
 (SEE 3 ON A)
 (SEE TOPOF C IS 2)
 (SEE 2 ON C)
 (C ISA PEG)
 (B ISA PEG)
 (A ISA PEG)
 (GOAL (PEG A))
 (GOAL 3 ON A)
 (GOAL (PEG B))
 (GOAL 1 ON B)
 (GOAL (PEG C))
 (GOAL 2 ON C)
 (ATTENDED (TOPGOAL (PEG A 3) (PEG B 1) (PEG C 2)))
 (Q3))
#
(undribble)
```